Mobile / Wireless Applications mit J2ME am Beispiel eines Instant-Messaging-Clients

Diplomarbeit
im Fach Softwareentwicklung
Studiengang Medieninformatik
der
Fachhochschule Stuttgart –
Hochschule der Medien

Tobias Frech

Erstprüfer:	Prof. Dr. Fridtjof Toenniessen
Zweitprüfer:	Dr. Stephan Rupp

Bearbeitungszeitraum: 27. 9. 2002 bis 26. 1. 2003

Stuttgart, Januar 2003

Danksagung

Ich möchte mich herzlich bei Professor Toenniessen für das Vertrauen und die gute Zusammenarbeit bedanken.

Großer Dank gilt auch Stephan Rupp, der meine Diplomarbeit betreut, mich unterstützt und stets mit gutem Rat mir geholfen hat.

Bei Ansgar Gerlicher will ich mich bedanken, für die Zeit, in der er mir zur Seite stand und mich beraten hat.

Ich bin meinen Freunden und Bekannten dankbar, für die tolle Unterstützung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AIM	AOL Instant-Messanger
API	Application Program Interface
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
CHTML	Compact HTML
CSD	Circuit-Switched-Data
DECT	Digital European Cordless Telecommunication
DOM	Domain Object Model
GCF	Generic Connection Framework
HSCSD	High-Speed-Circuit-Switched-Data
HTML	Hypertext Markup Language
IEEE	Institute of Electrical and Electronical Engineers
ICQ	„I seek you“
IM	Instant-Messaging oder Instant-Message
IP	Internet Protocol
IrDA	Infrared Data Association
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Mirco Edition
J2SE	Java 2 Standard Edition
JAD	Java Application Descriptor
JAR	Java Archiv
JDK	Java Development Kit
JNI	Java Native Interface
JSR	Java Specification Request
KVM	Kilo Virtual Machine
LAN	Local Area Network
LCDUI	Liquid Crystal Display User Interface
MDA	Mobile Digital Assistant
MID	Mobile Information Device

MIDP	Mobile Information Device Profile
MSM	Microsoft-Messanger
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
OTA	Over-The-Air Provisioning
PAN	Personal Area Network
PDA	Personal Digital Assistant
RMS	Record Management System
SMS	Short Message Service
SDK	Software Development Kit
UMTS	Universal Mobile Telecommunication System
VM	Virtual Machine
VoIP	Voice over IP
W3C	World Wide Web Consortium
WAP	Wireless Application Protokoll
WBXML	WAP Binary XML Content Format
WLAN	Wireless Local Area Network
WML	Wireless Markup Language
WTK	Wireless Toolkit
XML	Extensible Markup Language

1. Einleitung

1.1. Zielsetzung der Diplomarbeit

Diese Diplomarbeit gibt eine Einsicht in das Themengebiet „Mobile Applications“ (mobile Software-Anwendungen). Es wird dabei auf die einzelnen, neuen Technologien eingegangen und aufgezeigt welche Möglichkeiten in diesem Bereich entstehen und wie stark unser Leben in den nächsten Jahren davon geprägt werden kann.

Im praktischen Teil der Diplomarbeit habe ich eine mobile Anwendung, einen Instant-Messaging-Client, in J2ME entwickelt. Dieser Instant-Messaging-Client unterstützt das offene, XML-basierte Jabber-Protokoll. Auf die Entwicklung dieses Clients wird hier später genauer eingegangen (5. Kapitel). In dieser Arbeit wird der Begriff „Instant-Messaging“ – wie allgemein üblich – mit „IM“ abgekürzt.

Zuerst wird erklärt worin der Unterschied zwischen herkömmlichen Anwendungen und mobilen Anwendungen besteht und wie die aktuelle Marktsituation ist. Dann geht die Arbeit auf die verschiedenen Technologien ein, zeigt einen Überblick über die verschiedenen Programmiersprachen und Übertragungstechniken. Das J2ME Framework in einem eigenen Kapitel unter die Lupe genommen und die Funktionsweise anhand von einfachen Beispielen sowie typische Probleme und deren Lösungsansätze erklärt. Zum Schluß werden noch einige Möglichkeiten aufgezeigt, was mobile Anwendungen heute schon alles leisten können.

1.2. Persönliche Motivation

Seit mehr als 6 Jahren beschäftige ich mich intensiv mit Softwareentwicklung für Internet-Anwendungen. Daher habe ich schon zu Beginn der Diplomarbeit gute Kenntnisse in Client-Server-Technologien und in der Programmiersprache Java. Für das Thema „Mobile Applications“ habe ich mich entschieden, weil ich es sehr faszinierend finde, was heute in diesem Bereich schon möglich ist und ich fest davon überzeugt bin, dass der Einfluss dieser Technologie unsere Zukunft stark prägen wird.

Es gab für mich aber auch noch einen anderen Grund: Einer der zur Zeit erfolgreichsten mobilen Services ist SMS. SMS ist dem IM sehr ähnlich. In beiden Fällen werden Kurznachrichten zwischen zwei Endgeräten ausgetauscht. IM hat aber zudem meist noch weitere Dienste, wie zum Beispiel die Möglichkeit zu sehen, ob ein Freund gerade online ist oder nicht. In komplexeren IM-Systemen ist sogar Video-Telefonie (mit Audio/Video-Streaming) möglich.

Das enorme Potential von mobilen IM-Systemen und die Faszination dieser Technik sind die Gründe, warum ich mich dazu entschlossen habe, in meiner Diplomarbeit, einen mobilen IM-Client zu entwickeln.

2. Mobile Applications

2.1. Was sind Mobile Applications

„Mobile Applications“ ,oder oft auch „Wireless Applications“ genannt, werden im Vergleich zu herkömmlichen Anwendungen für mobile Endgeräte entwickelt. Mobile Endgeräte können dabei programmierbare Mobiltelefone, Smartphones, PDAs (Personal Digital Assistent), MDAs (Mobile Digital Assistent) oder auch mobile Kleincomputer wie SimPads und auch Laptops sein. Die mobilen Endgeräte sind klein, sehr flexibel und räumlich unabhängig (daher kommt der Name „Mobile“) und haben entweder keine oder meist eine kabellose Verbindung zu anderen Geräten (daher der Name „Wireless“). Das mobile Endgerät ergibt zusammen mit der passenden mobilen Anwendung eine „Mobile Solution“ oder „Wireless Solution“, also ein System mit sehr flexiblen und vor allem ortsunabhängigen Eigenschaften und Möglichkeiten.

Dass man Anwendungen für Laptops oder PDAs entwickelt, ist schon lange nicht mehr revolutionär. Aber der Gedanke, dass bald jedes Mobiltelefon – umgangssprachlich auch Handy genannt – mit speziellen Anwendungen zu einem universellen Multifunktionsgerät werden kann, eröffnet ungeahnte Möglichkeiten. Das Mobiltelefon ist klein, kostengünstig und sehr weit verbreitet. Die meisten der neuen Mobiltelefone sind mit J2ME programmierbar und eignen sich somit optimal für mobile Anwendungen für den Massenmarkt. Laut einer Studie von Gartner Research[1] wird es in den nächsten Jahren bei den programmierbaren Endgeräten mehr Mobiltelefone als stationäre Geräte, wie zum Beispiel PCs, geben. Dadurch wird erst die große Bedeutung dieses noch jungen Geschäftsfeldes klar.

Vor einigen Jahren, als der erste programmierbare PDA von der Firma Palm erfolgreich auf den Markt gebracht wurde, haben viele Entwickler weltweit Anwendungen für diese Geräte geschrieben und somit den Erfolg der PDAs erst möglich gemacht. Jeder Endbenutzer konnte seinen PDA um die gewünschten Funktionen erweitern. Heute gibt es enorme Anzahl von Anwendungen für PDAs wie Palm und PocketPCs. Nach der Einführung von J2ME als plattformunabhängige Programmiersprache für Mobiltelefone und andere mobile Endgeräte steht diesen Geräten die gleiche Zukunft bevor. Jetzt schon gibt es hunderte Spiele und einige Tools, die man sich auf sein javafähiges Mobiltelefon downloaden kann.

[1] [JS] S. 20

2.2. Die mobilen Endgeräte

Wie auch schon in der Einleitung erwähnt wurde, gibt es verschiedene Gruppen von Geräten, die für mobile Anwendungen geeignet sind. Die folgenden Tabellen zeigen einen Überblick über die verschiedenen Gruppen:

Es gibt noch weitere Geräte, die mit J2ME programmiert werden können. Dazu zählen bestimmte Set-Top-Boxen für den Empfang von digitalem Fernsehen, Haushaltsgeräte wie Waschmaschinen, Kaffeemaschinen usw.. Theoretisch ist jedes Gerät durch die Integration von J2ME beliebig programmierbar. Da diese Geräte aber derzeit noch nicht auf dem Markt verfügbar sind, werden diese nicht in den Tabellen aufgeführt.

Tabelle 1: Überblick der verschiedenen mobilen Endgerätetypen

Mobiltelefon / Smartphone	
Zielgruppe	Massenmarkt, Jugendliche (Spielehandy) bis Geschäftsleute (Businesshandy)
Verbreitung	Sehr große Verbreitung
Preis	100,- bis 1.000,- €
Betriebssystem	Properitär oder SymbianOS (früher EPOC)
Programmiersprachen	Java (J2ME), C++, WML, i-mode
CPU	Langsame 16-bit oder 32-bit CPU
Speicher	Programmspeicher meist < 50kB, in Ausnahmefällen bis zu 500kB. Bei Smartphones bis zu 4MB
Kabellose Verbindungsarten	SMS, GSM, GPRS, IrDA, Bluetooth, evtl. bald UMTS
Extras	Kameras, Organizer, Emailclient

PDA / MDA	
Organizer mit (MDA) oder ohne (PDA) eingebautem Telefon	
Zielgruppe	Geschäftsleute, Spezialanwendungen
Verbreitung	Mittlere Verbreitung
Preis	400,- bis 1.000,- €
Betriebssystem	WinCE oder PalmOS
Programmiersprachen	Java (J2ME), C++, C, VisualBasic, WML, HTML
CPU	Schnellere 32-bit CPU
Speicher	Programmspeicher ca. 32MB bis 64MB
Kabellose Verbindungsarten	PDA: Bluetooth, WLAN, IrDA und mit AddOn-Cards auch SMS, GSM, GPRS, evtl. bald UMTS MDA: SMS, GSM, GPRS, IrDA, Bluetooth, evtl. bald UMTS
Extras	Große Farbdisplays, Kameras, Organizer, Emailclient, Touchscreen, Office-Software, Audio/Video-Player, zum Teil erweiterbar mit CF-Karten oder PCMCIA-Karten

SimPad	
Zielgruppe	Privat- und Geschäftsleute, Spezialanwendungen
Verbreitung	Geringe Verbreitung
Preis	1.000,- bis 2.000,- €
Betriebssystem	WinCE
Programmiersprachen	Java (J2ME), C++, C, VisualBasic, HTML
CPU	Schnellere 32-bit oder 64-bit CPU
Speicher	Programmspeicher ca. 32MB
Kabellose Verbindungsarten	IrDA, erweiterbar durch PCMCIA-Karten: WLAN, SMS, GSM, GPRS, Bluetooth, evtl. bald UMTS
Extras	Farbdisplays, Kameras, Organizer, Emailclient, Touchscreen, Office-Software, Audio/Video-Player

(Fortsetzung der Tabelle auf der nächsten Seite)

Laptop	
Zielgruppe	Meist Geschäftsleute, Aussendienstmitarbeiter
Verbreitung	Große Verbreitung
Preis	1.000,- bis 3.000,- €
Betriebssystem	z.B. WinXP oder Linux
Programmiersprachen	Java (J2SE), C++, C, VisualBasic, HTML und viele Andere
CPU	Schnellere 32-bit oder 64-bit CPU
Speicher	Programmspeicher ca. 32MB bis mehrere hundert MB
Kabellose Verbindungsarten	WLAN, IrDA, Bluetooth, erweiterbar durch PCMCIA-Karten: WLAN, SMS, GSM, GPRS, evtl. bald UMTS
Extras	Farbdisplays, Kameras, Organizer, Emailclient, Touchscreen, Office-Software, Audio/Video-Player

In der Zeit während der Erstellung dieser Diplomarbeit haben sich die mobilen Endgeräte stark weiterentwickelt. Zu Beginn waren nur wenige javafähige Mobiltelefone verfügbar; jetzt (2 Monate später) unterstützt nahezu jedes Mobiltelefon, das neu auf den Markt kommt, J2ME. Zudem wird Bluetooth in immer mehr Endgeräten eingebaut.

Auf der Suche nach dem optimalen Endgerät muss man sich genau überlegen, welche Zwecke das Gerät im Gesamtsystem erfüllen soll. Dann wird schnell klar, ob ein PDA auf Grund seiner kompakten Ausmaße und des dafür doch relativ großen und hochauflösenden Display oder ein Mobiltelefon, welches in viel größeren Stückzahlen auf dem Markt vertreten ist das richtige mobile Endgerät ist. Das optimale Endgerät für alle Zwecke gibt es noch nicht. Folgende Tabelle zeigt die Vor- und Nachteile bzw. Grenzen der einzelnen Gerätetypen auf.

Tabelle 2: Vor- und Nachteile der verschiedenen mobilen Endgerätetypen

Gerätetyp	**Vorteile**	**Nachteile**
Mobiltelefon / Smartphone	+ Klein + Kostengünstig + Große Verbreitung + Telefon und programmierbares Endgerät in einer Einheit	- Kleines Display - Wenig Speicher - Langsame CPU - Umständliche Bedienung
PDA / MDA	+ Großes Display + Einfachere Bedienung durch Touchscreen + Leistungsfähig + MDA: Telefon und programmierbares Endgerät in einer Einheit	- PDA: Telefonfunktion nur mit Erweiterungsmodulen - Als Mobiltelefon zu groß und zu schwer - Hoher Preis
SimPad	+ Sehr großes Display + Einfachere Bedienung durch Touchscreen + Leistungsfähig	- Zu schwer und zu groß als richtig portables Gerät - Hoher Preis
Laptop	+ Sehr großes Display + Einfachere Bedienung durch richtige Tastatur und Mousepad + Leistungsfähig wie ein Desktop-PC	- Zu schwer und zu groß als richtig portables Gerät - Hoher Preis

Die Mobiltelefone und Smartphones werden immer leistungsfähiger und übernehmen mehr und mehr die Funktionalitäten von PDAs. Diese Telefone sind für mobile Anwen-

dungen sehr interessant, da diese Geräte in sehr großen Stückzahlen auf dem Markt verfügbar sind, von den Mobilfunkprovidern subventioniert werden und klein genug für den täglichen Gebrauch und somit ständige Begleiter sind. Vermutlich werden in ein paar Jahren nur noch leistungsfähige Smartphones als tägliche digitale Begleiter für Kommunikation und Informationsaustausch, sowie Organisation, Komfort, Sicherheit und zur Fernsteuerung auf dem Markt verfügbar sein und somit die PDAs völlig verdrängen. Da Mobiltelefone in der Zukunft eine so große Rolle spielen werden liegt der Fokus der Diplomarbeit auf dieser Gerätegruppe.

2.3. Bereits verfügbare Software

Für PDAs gibt es fast für jeden Zweck die passende Software. So kann man von Routenplanung über Bahnauskunft bis hin zu Videosoftware oder Office-Anwendungen alles finden. Sogar die Computerspieleindustrie entdeckt auch langsam den PDA-Markt.

Bei Mobiltelefonen sieht es schon ganz anders aus. Der Großteil der verfügbaren Software sind Spiele. Ein Grund dafür wird sicher auch Vodafone's stark beworbener Spieleservice „Load-A-Game“ sein. Als eine der besonderen Highlights kann man, die für das Nokia 7650 portierte Version des Firstperson-Shooter Klassikers, DOOM aufzählen. Hier wird auf dem Mobiltelefon eine 3D-Umgebung flüssig in Echtzeit dargestellt. Business-Software wie zum Beispiel Textverarbeitung und andere Office-Anwendungen, Emailclients, Browser, Terminkalender, Chats oder ähnliches findet man noch nicht so oft. Mit dem Einzug von J2ME auf fast allen Mobiltelefonen kann sich dies schlagartig ändern.

3. Technologien

In diesem Kapitel werden die verschiedenen Technologien vorgestellt, die zur Erstellung von mobilen Lösung verwendet werden können. Dabei werden zuerst die Programmiersprachen und dann die verschiedenen Übertragungstechnologien betrachtet.

3.1. Programmiersprachen

Es gibt verschiedene Programmiersprachen um Anwendungen für mobile Endgeräte zu entwickeln. Je nach Anforderung der Lösung (Stand-Alone, Peer-to-Peer, Client-Server, usw.) und Endgerät wählt man die Programmiersprache aus. Die Programmiersprachen unterteilen sich in zwei verschiedene Kategorien: Applikationen (eigenständige Anwendungen) und auf Browser basierende Lösungen.

3.1.1. Applikationen

Für Anwendungen mit einer eigenen komplexen Programmlogik entwickelt man Applikationen. Diese sind auch ohne Netzverbindung oder einem Server lauffähig. In diese Kategorie fallen zum Beispiel Spiele, Peer-to-Peer-Anwendungen, Organizer-Programme und viele mehr. Die meisten Mobiltelefone unterstützen J2ME und können somit Java-Anwendungen ausführen. Bei einigen Smartphones oder anderen Endgeräten können Applikationen auch in C++ oder C entwickelt werden. Für WinCE-Geräte, wie PocketPCs, steht zusätzlich auch VisualBasic zur Verfügung.

Mit dem J2ME-Framework ist Java momentan die am häufigsten unterstützte Programmiersprache bei mobilen Endgeräten. Applikationen für J2ME-kompatible Endgeräte nennt man MIDlets (analog zu Applets) wobei MID für Mobile Information Device steht. Der große Vorteil von Java ist die Plattformunabhängigkeit. Auf J2ME wird im 4. Kapitel noch genauer eingegangen.

3.1.2. Browserbasierte Systeme

Client-Server-Systeme kann man wahlweise mit einer eigenständigen Anwendung oder einem browserbasierten System realisieren. Bei Client-Server-Systemen entscheidet sich die Wahl der Programmiersprache durch die Anforderung des Benutzerschnittstelle. Für einfache Informationsdarstellung verwendet man browserbasierte Sprachen für Anwendungen mit einer anspruchsvolleren Logik der Benutzerschnittstelle fällt die Wahl auf Applikationen.

3.1.2.1. HTML

Für die Informationsdarstellung im Internet wird meist das World-Wide-Web genutzt welches als Grundlage die Dokumentensprache HTML nutzt. Da aber die HTML-Seiten

heute mit Multimediainhalten für große Bandbreiten und leistungsfähige Endgeräte optimiert werden, eignet sich diese Sprache nicht mehr für den mobilen Einsatz. Deshalb wurde für diese Zwecke eine Abwandlungen von HTML wie WML oder cHTML entwickelt.

3.1.2.2. WML

WML[2] ist eine XML-Sprache und wurde als Seitenbeschreibungssprache für Mobiltelefone entwickelt. WML-Seiten sind kompakt, enthalten nur wenig Grafik und verfügen über umfassende WMLScript-Funktionen. WMLScript ist mit dem JavaScript bei HTML-Seiten verwandt. Die Grafiken in WML-Seiten sind nur schwarz-weiss (ohne Graustufen).

WML unterscheidet zwischen Decks und Cards. Eine WML-Datei ist ein Deck, bestehend aus einer oder mehreren Cards. Eine Card stellt eine Darstellungsseite auf dem Mobiltelefon dar. Der Benutzer kann ohne Datentransfer durch die Cards eines Decks navigieren.

Mit diesen Möglichkeiten können schon relativ gute Client-Server-Systeme für die mobile Informationsdarstellung entwickelt werden. Jedoch ist für die Entwickler solcher Seiten eine gewisse Einarbeitungszeit notwendig.

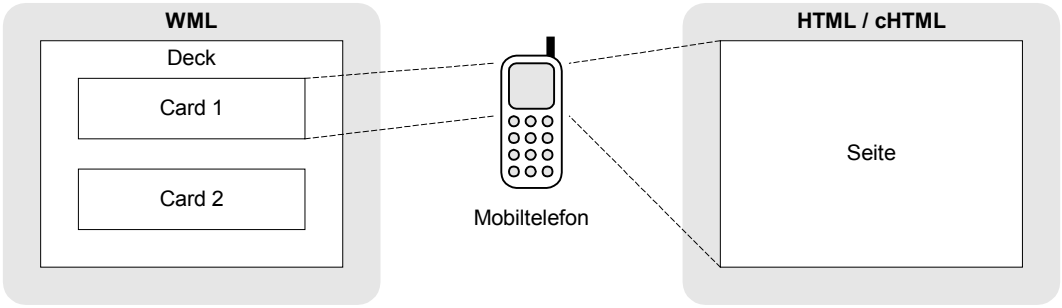

Abbildung 1: Darstellung von WML-, HTML- und cHTML-Seiten im Vergleich

3.1.2.3. cHTML und i-mode

cHTML[3] ist eine HTML-Variante, die speziell für die Informationsdarstellung auf kleinen mobilen Endgeräten wie Mobiltelefone optimiert wurde. In cHTML-Seiten können keine Frames, Tabellen oder JPG-Grafiken verwendet werden. Auch Stylesheets, Hintergrundfarben oder -bilder werden nicht unterstützt. Neu dazugekommen ist die Möglichkeit Hyperlinks mit einer Telefontaste zu verbinden. Somit kann der Benutzer auch mit einem Mobiltelefon schnell und einfach navigieren. Weiter gibt es das aus dem Internet Explorer bekannte <marquee>-Tag. Mit diesem Tag können Laufschriften erstellt werden. Der Vorteil von cHTML im Vergleich zu WML liegt darin, dass die Seiten schneller und einfacher erstellt werden können, da jeder HTML-Programmierer ohne großen

[2] [WML]
[3] [CHTML] und [IMODE]

Mehraufwand cHTML-Seiten entwickeln kann. Zudem unterstützt cHTML im Vergleich zu WML 256 Farben und GIF-Grafiken.

Im Zusammenhang mit i-mode wird cHTML oft iHTML genannt. iHTML steht für „i-mode compatible HTML“. i-mode bietet im Vergleich zu den anderen Übertragungstechnologien ein Billingsystem für Content-Anbieter, mit dem Kosten direkt über den Mobilfunkprovider abgerechnet werden.

3.2. Übertragungstechnologien

Zum Datenaustausch zwischen mobilen Endgeräten oder zwischen mobilem Endgerät und dem Internet oder anderen Netzwerken gibt es verschiedene Übertragungstechnologien.[4]

3.2.1. IrDA

Über die Infrarotschnittstelle in vielen Endgeräten können Daten mit einer Übertragungsgeschwindigkeit von 115,2 kBit/s (bei SIR – Standard Infrared) bis zu 16 MBit/s (bei VFIR – Very Fast Infrared) und einer Reichweite von 1 bis 2 Metern übertragen werden. Die bekannteste IrDA-Anwendung ist die Fernbedienung. Der Datenaustausch erfolgt über das sogenannte Tiny Transport Protocol. Auf dieses Protokoll setzen drei weitere auf: IrLAN für den Zugang zu lokalen Netzwerken, IrOBEX für den Datenaustausch und IrCOMM als Simulation serieller oder paralleler Schnittstellen. Für eine Infrarotverbindung muss direkter Sichtkontakt zwischen den Geräten bestehen. Die größten Nachteile von IrDA sind zum einen die hohe Störanfälligkeit bei Sonnenlicht, die geringe Reichweite und der notwendige Sichtkontakt zwischen den Geräten. Daher ist diese Übertragungstechnik nur bedingt zu empfehlen. Bluetooth wird IrDA sicher irgendwann ablösen, weil bei einer Bluetoothübertragung diese Probleme nicht bestehen.

3.2.2. DECT

DECT ist ein Funkstandard, der hauptsächlich Verwendung bei kabellosen Telefonen findet. Es ist nur für Punkt-zu-Punkt-Verbindungen zwischen einer Basisstation und einem Mobilteil ausgelegt. Eine direkte Verbindung zwischen zwei Mobilteilen ist nicht möglich. Mit DECT sind Übertragungsgeschwindigkeiten von bis zu 552 kBit/s bei einer Reichweite von 40 Meter (innerhalb von Gebäuden) und 350 Meter (im Freien) möglich. Der große Vorteil von DECT ist die niedrige Störanfälligkeit, da für diese Technik ein reserviertes Frequenzband und ein vorgeschriebenes Zugriffsverfahren existieren. Ein Nachteil jedoch ist, dass weltweit verschiedene Frequenzen verwendet werden.

[4] [ECIN], [GSM] und [CHIP]

3.2.3. SMS

SMS ist keine Übertragungstechnologie, sonder ist ein Service zum versenden von 160 Zeichen langer Textnachrichten zwischen verschiedenen Mobiltelefonen. Dieser Service wird hier trotzdem aufgeführt, da zum einen dieser Serivce ein eigenes Abrechnungsmodel hat und zum anderen dieser Service auch für die Kommunikation zwischen Endgeräten verwendet werden kann. Manche Mobiltelefonhersteller bieten sogar dafür eine eigene SMS-API an. Dieser Service ist momentan die erfolgreichste Mobile Anwendung auf dem Markt. Monatlich werden allein in Deutschland über 1 Milliarde SMS verschickt.[5] In SMS können auch Bildinformationen (z.B. Betreiberlogos) oder Klingeltöne sowie Konfigurationsinformationen oder auch andere Daten verschickt werden. Mit Hilfe von sogenannten SMS-Gateways oder GSM-Modems kann man auch diese Kurznachrichten direkt vom PC verschicken oder dort auch empfangen. SMS werden hauptsächlich zum Nachrichtenaustausch zwischen Menschen verwendet, könnten aber auch zum versenden von Steuerungsdaten, Highscoredaten bei Spielen oder zu anderen Zwecken genutzt werden. Die Preise für eine SMS liegen zwischen 5 und 30 cent (je nach Anbieter und Stückzahlen).

3.2.4. GSM: CSD/HSCSD

Das Global System for Mobile Communication – kurz GSM – ist das heute weltweit am meisten genutzte Mobiltelefonsystem. Die Mobiltelefone mit GSM-Standard haben eine Übertragungsrate von 9,6 kBit/s. Eine GSM-Verbindung (CSD) ist leitungsorientiert und wird nach Zeit berechnet. Durch das Bündeln von mehreren Kanälen wird eine größere Übertragungsgeschwindigkeit von bis zu 43,2 kBit/s erreicht. Diese Technik nennt man HSCSD. Die Kosten für eine Online-Minute liegt bei ca. 10 bis 20 cent.

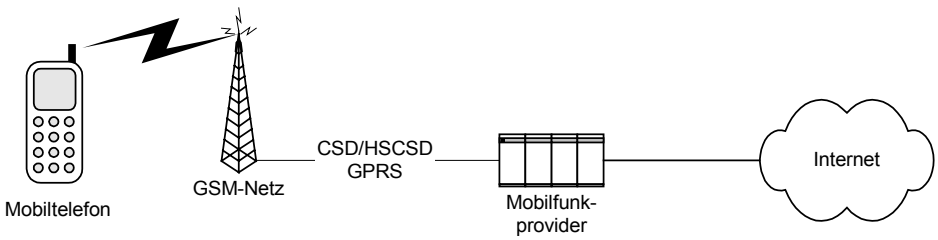

Abbildung 2: Schema für die Verbindung zum Internet aus dem GSM-Netz

3.2.5. GPRS

GPRS ist ein erweiterter GSM-Standard, mit dem eine paketorientierte Verbindung möglich ist. Die Verbindungskosten werden nicht nach Zeit, sondern nach Transfervolumen berechnet. Mit GPRS ist eine theoretische Geschwindigkeit von bis zu 115 kBit/s möglich. In der Realität wird das nie erreicht, weil alle Mobiltelefone in einer Funkzelle sich diese Bandbreite teilen müssen. In der Praxis wird mit einer realistischen Größe

[5] [SMS]

von 28 kBit/s gerechnet. GPRS ist heute in fast allen Netzen flächendeckend verfügbar. Durch die Einführung von GPRS ist es zum ersten mal möglich mobile Anwendungen zu entwickeln, die einen ständige Verbindung zum Internet benötigen. Die Kosten für 10 kByte liegen zwischen 1 und 10 cent.

3.2.6. UMTS

UMTS ist der Mobilfunkstandard der 3. Generation. Hohe Übertragungsgeschwindigkeiten von bis zu 2 MBit/s sind damit möglich. Auch bei UMTS muss die Bandbreite innerhalb einer Funkzelle zwischen den aktiven Benutzern geteilt werden. Interaktive Breitbanddienste über öffentliche Funkzellen sind grundsätzlich schwierig bzw. teuer. UMTS ermöglicht aber gegenüber GSM und GPRS vor allen Dingen einen effektiven Umgang mit der Luftschnittstelle für Datendienste. Die Vorteile von UMTS sind folgende:

- Kürzere Reaktionszeiten für die Allokation von Ressourcen auf der Luftschnittstelle bei Bedarf
- Unterstützung unterschiedlicher Dienstgüteklassen für unterschiedliche Arten von Diensten (Conversational, Streaming, Interactive, Background)
- Kostengünstigere Angebote für Datendienste (Instant-Messaging-Clients können zum Beispiel in der billigsten Kategorie „Background“ laufen)

Da jedoch die ersten UMTS-Netze erst in einigen Jahren verfügbar sein werden und mit GPRS schon nahezu alles außer Videoübertragung möglich ist, wird auf diese Technologie hier nicht weiter eingegangen. Es gibt schon Test des UMTS Nachfolgers von NTT Docomo: Der Mobilfunk der 4. Generation soll 100 Megabits pro Sekunde unidirektional übertragen können.[6]

3.2.7. WLAN

Wireless-LAN oder WLAN ist eine Netzwerkverbindung über Funk bei der IP-Pakete, mit Übertragungsgeschwindigkeiten von bis zu 54 MBit/s[7], übertragen werden können und ist unter dem Namen IEEE 802.11 weltweit als Standard bekannt. Von diesem Standard gibt es verschiedene Versionen, die sich in der technischen Umsetzung unterscheiden. Mit 802.11a und 802.11g ist eine Datenrate von 54 MBit/s, bei 802.11b nur von 11 MBit/s möglich. Die verschiedenen Versionen sind nur sehr eingeschränkt zueinander kompatibel, da sich die Frequenz und auch das Modulationsverfahren unterscheiden. Die Reichweite beträgt in Gebäuden ungefähr 30 Meter und im Freien ungefähr 300 Meter. Es gibt zwei verschiedene Betriebsarten: Im Ad-hoc-Modus kommunizieren alle Endgeräte in einem Peer-2-Peer-Netzwerk direkt miteinander. Im Infrastruktur-Modus erfolgt die Kommunikation über einen zentralen Access-Point. Jeder ist

[6] [FTD]
[7] [ARTEM]

in der Lage ein solches Netzwerk mit der entsprechenden Hardware, die im guten Fachhandel erhältlich ist, aufzubauen. Möchte man zum Beispiel einen mobilen Internetzugang anbieten ist ein sogenannter Access Point notwendig bei dem sich das mobile Endgerät anmelden kann. Die Verbindung ist prinzipiell kostenfrei. Immer mehr Restaurants oder Cafés, sowie Bahnhöfe, Flughafen, Einkaufszentren oder Hotels bieten ihren Kunden gegen Gebühr einen WLAN Zugang mit Internetverbindung an. Die Kosten liegen hier bei ca. 2 bis 5 Euro je Stunde. In manchen Bereichen von Großstädten gibt es schon eine fast flächendeckende WLAN-Versorgung, die teilweise durch private oder auch kommerzielle Access-Points besteht. Zahlreiche Fachleute sehen daher WLAN als ernst zu nehmende Konkurrenz für UMTS, da hier keine teuren Lizenzgebühren refinanziert werden müssen. Der große Nachteil von öffentlichen, kommerziellen WLANs ist die Abrechnung, die meist nur über Prepaid-System realisiert ist. Für jeden Anbieter benötigt man eine eigene Prepaid-Karte. Bei UMTS kann alles zentral über den Mobilfunkprovider abgerechnet werden.

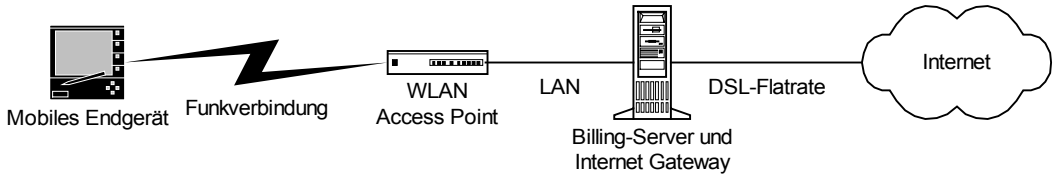

Abbildung 3: Schema für die Verbindung zum Internet über ein WLAN

3.2.8. HomeRF

HomeRF ist eine Kombination von DECT und WLAN. Somit steht ein zuverlässiger Sprachdienst und ein schneller Datendienst zur Verfügung. Mit dieser Technik ist eine Datenübertragung von bis zu 10 MBit/s möglich. Die paketvermittelte Datenübertragung ist sowohl in einem Peer-to-Peer-Netzwerk als auch über einen Control-Point (CP) möglich. Der Control-Point entspricht dem Access-Point bei WLAN. Theoretisch müsste HomeRF ein voller Erfolg sein. Da aber diese Technologie nicht von der Industrie (mit entsprechender Hardware) unterstützt wird löst sich die Arbeitsgruppe HomeRF und somit die Weiterentwicklung dieser Technologie auf.[8]

3.2.9. Bluetooth

Bluetooth ist ein Kurzstreckenfunkstandard für PANs (Personal Area Network) und wurde in erster Linie zum kabellosen verbinden von lokalen Geräten wie Drucker und PC entwickelt. Diese Technologie ist schon seit 3 Jahren auf dem Markt und nimmt so langsam Einzug in Mobiltelefonen (kabellose Freisprecheinrichtung, Multiuserspiele), PDAs (Datensynchronisation) und anderen Geräten. Auch als mobilen Internetzugang kann Bluetooth verwendet werden. Mit einem Palm PDA ist das Surfen schon in Frazer

[8] [HEISE1]

Coffeebars in Frankfurt möglich[9]. Der Durchbruch dieser Technologie wurde schon mehrmals vorausgesagt, ist aber bis jetzt noch nicht wirklich erfolgt. Die Nutzung ist, wie beim WLAN, auch kostenfrei. Lediglich müssen Hardwarehersteller ihre Produkte kostenpflichtig zertifizieren lassen. Mit Bluetooth kann eine theoretische Übertragungsgeschwindigkeit von bis zu 1MBit/s erreicht werden. Diese Werte werden aber in der Praxis nicht erreicht. Im asymmetrischen Verfahren sind 721 kBit/s in die eine und 57,6 kBit/s in die andere Richtung möglich. Symmetrisch werden in beide Richtungen 432,6 kBit/s übertragen. Bei Bluetooth sind Point-to-Point, sowie Point-to-Multipoint-Verbindungen möglich.

Bluetooth wird in 3 Klassen unterteilt:

- **Klasse 1**: Reichweite 1-5 Meter, 1 mW (0 dBm) Sendeleistung
- **Klasse 2**: Reichweite 10-20 Meter, 2,5 mW (4 dBm) Sendeleistung
- **Klasse 3**: Reichweite bis zu 100 Meter, 100 mW (20 dBm) Sendeleistung

Die meisten Bluetooth-Module verwenden Klasse 2. In Verbindung mit Mobiltelefonen kann Bluetooth zur Steuerung von Haushaltsgeräten, für personalisierte Anwendungen, für Sicherheitstechnik, für kabellose Headsets und für viele weitere Zwecke genutzt werden. In meinen Augen wird die Kombination von Mobiltelefon und Bluetooth endlich den Durchbruch dieser Technologie bringen.

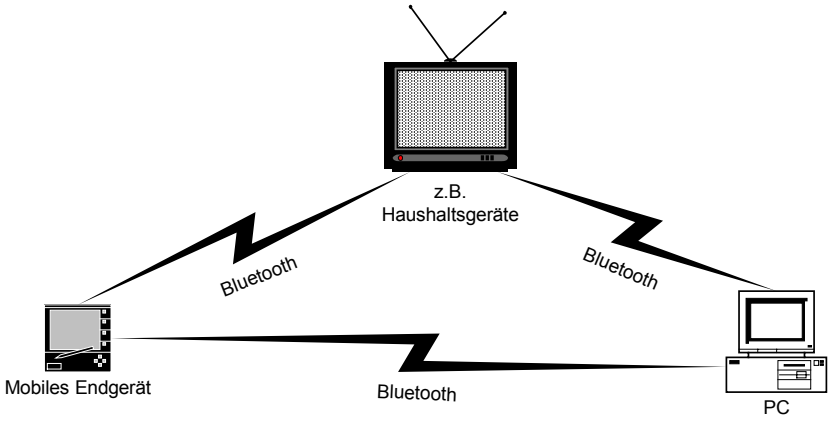

Abbildung 4: Bluetooth-Kommunikation zwischen verschiedenen Endgeräten

3.3. Zwischenfazit

Die meisten Mobiltelefone, die momentan erhältlich sind, unterstützen J2ME und GPRS. Mit diesen beiden Technologien können heute schon leistungsfähige mobile Anwendungen für den Massenmarkt entwickelt werden. Egal ob es dabei um ein datenbankgestütztes Client-Server-System oder um ein Stand-Alone-Spiel handelt. Die nächste Mobiltelefongeneration (ab Mitte 2003) wird zudem noch Bluetooth unterstüt-

[9] [HEISE2]

zen. Meiner Meinung nach wird das Mobiltelefon damit schon bald als leistungsfähiges und vor allem kostengünstiges Endgerät für die vielseitigsten Anwendungszwecke entdeckt werden.

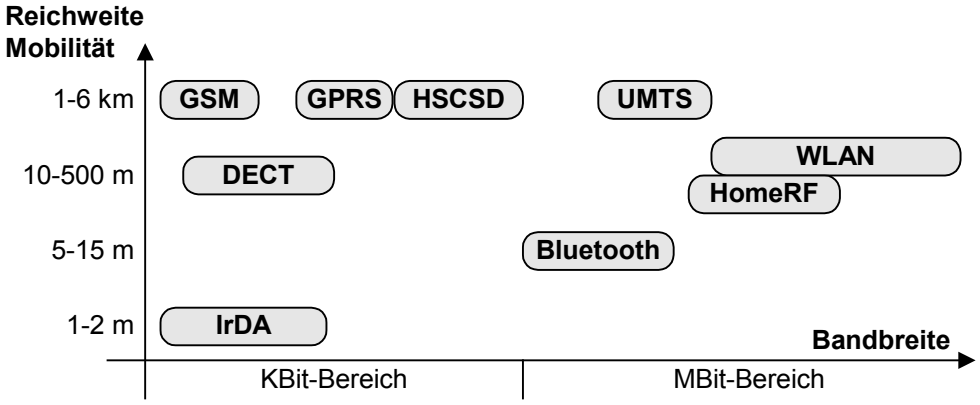

Abbildung 5: Übertragungstechnologien im grafischen Vergleich

Tabelle 3: Tabellarischer Vergleich der verschiedenen Übertraungstechnologien

	Bandbreite in Bit/s	**Frequenz**	**Reichweite in Meter**	**Verfügbarkeit**	**Traffic-Kosten**
IrDA	115,2K bis 16M	850-900 nm	1 bis 2	--	--
DECT		1880-1900 MHz	40 bis 350	--	--
CSD (GSM)	9,6K	900, 1800 und 1900 MHz	---	Deutschlandweit	10-20 cent/Min
HSCSD (GSM)	28,8K bis 43,2K	900, 1800 und 1900 MHz	---	Deutschlandweit	10-20 cent/Min
GPRS	28K	900, 1800 und 1900 MHz	---	Deutschlandweit	1-10 cent/10kB
WLAN	11M bis 54M	2,4 GHz oder 5 GHz	30 bis 300	In Großstädten, Flughäfen, Bahnhöfe, Hotels, Restaurants	Kostenlos oder 2-5 €/Std.
HomeRF	10M	2,4 GHz	50	--	--
Bluetooth	1M	2,4 GHz	1 bis 100	Als Internetzugang in einem Frankfurter Coffeebar	Kostenlos oder 2-5 €/Std
UMTS	2M	1920-1980 und 2110-2170 MHz	---	Nicht verfügbar	Unbekannt

4. J2ME – Java 2 Micro Edition

J2ME[10] ist ein Java-Framework das speziell für mobile Endgeräte wie zum Beispiel Mobiltelefone entwickelt wurde. Dabei wurde beachtet, dass alle diese Geräte folgende Eigenschaften aufweisen:

- Langsamer Prozessor (geringe Performance)
- Wenig Speicher
- Zum Teil niedrige Übertragungsgeschwindigkeiten
- Begrenze Eingabe (keine oder nur eine kleine Tastatur)
- Schlechte Darstellungsmöglichkeiten (kleine Displays)
- Zum Teil begrenzte Stromleistung (Akku)

Die folgende Abbildung zeigt eine Übersicht über die verschiedenen Java Frameworks, die jeweilige verwendete VM, den verfügbaren Speicher und Prozessortyp sowie die typischen Endgeräte.

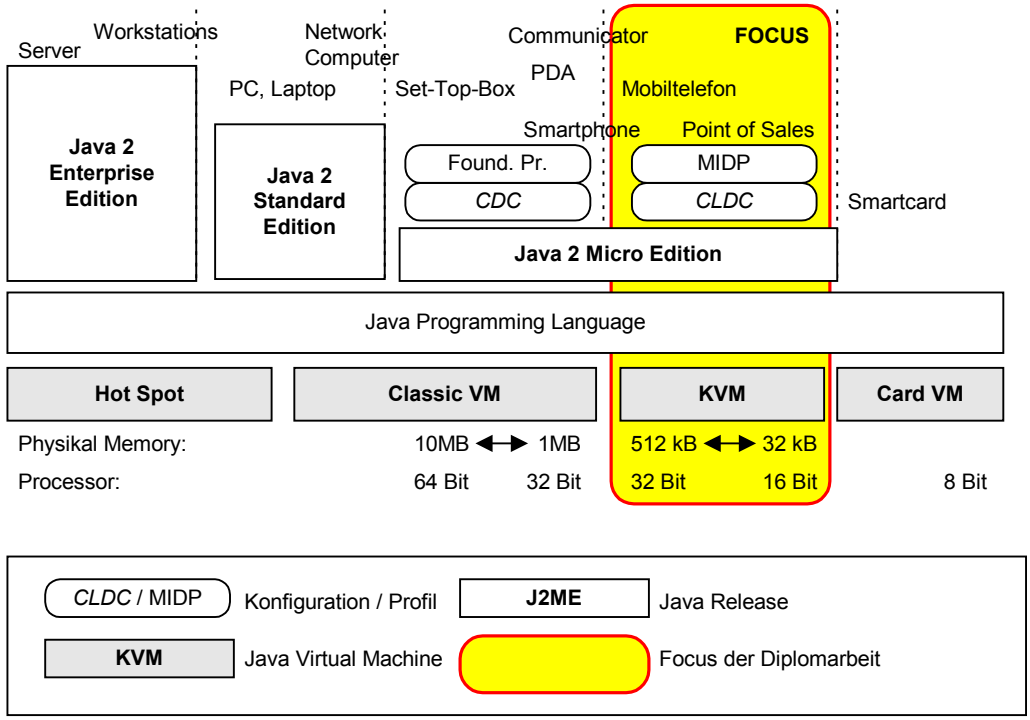

Abbildung 6: Java Releases mit jeweiliger VM und Endgeräte

[10] [SUN], [MIDP], [J2ME], [BHV] und [RIGGS]

4.1. Der Aufbau

Da nicht alle J2ME-Geräte die gleichen Leistungseigenschaften oder Restriktionen haben, aber trotzdem ein möglichst breites Spektrum an Endgeräten angesprochen werden soll, ist J2ME in zwei Komponenten unterteilt: Die sogenannte Konfiguration und das Profil. Die Konfiguration beschreibt einen universell verfügbaren, minimalen Funktionsumfang für eine allgemeine Geräteklasse, wie zum Beispiel Mobiltelefone. Die Konfiguration beschreibt die Sprachelemente, die unterstützt werden, die passende VM und die passende API. Für jede Konfiguration gibt es verschiedene Profile, die dann diese zu einer vollständigen Laufzeitumgebung erweitern.

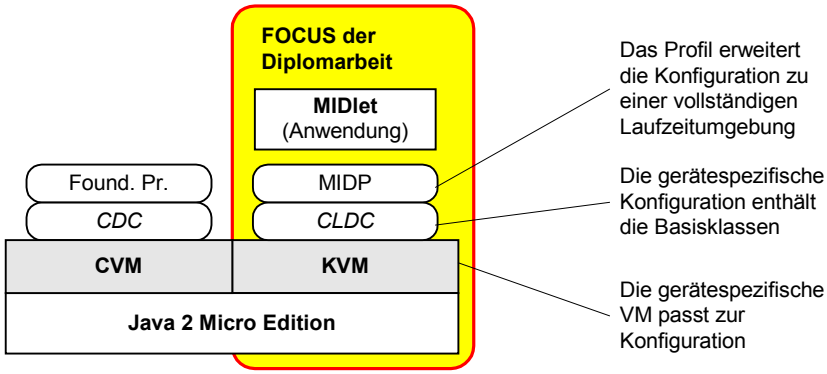

Abbildung 7: J2ME mit Konfiguration und Profil und MIDlet

Diese Diplomarbeit wird sich hauptsächlich mit der Konfiguration (CLDC) für Mobiltelefone und dem dazu passenden Profil (MIDP) auseinandersetzen. Die andere derzeit verfügbare Konfiguration und die weiteren Profile werden nur zur Vollständigkeit erwähnt.

Momentan gibt es zwei verschiedene Konfigurationen: CDC und CLDC. CDC steht für Connected Device Configuration und ist die typische Konfiguration für PDAs, Set-Top-Boxen oder andere Geräte mit folgenden Eigenschaften gedacht:

- 32 oder 64 Bit-CPU
- mindestens 2 MB Speicher für Java
- Zugang zu einem Netzwerk

Die CDC enthält nahezu alle J2SE-Klassen und arbeitet mit der normalen Java-VM. Das einzig derzeit verfügbare Profil ist das Foundation-Profil. In dieser Diplomarbeit wird nicht weiter auf diese Konfiguration, die VM und das Profil eingegangen, da für die heutige Generation von Mobiltelefonen nur die CLDC interessant ist. Die CDC enthält alle Funktionalitäten, Klassen und Interfaces der CLDC.

4.1.1. CLDC (Konfiguration)

Die Connected Limited Device Configuration, kurz CLDC, ist die übliche Konfiguration für Mobiltelefone und ähnliche Geräte mit sehr beschränken Eigenschaften:

- 16 oder 32 Bit-CPU
- 128kB oder mehr Speicher für Java
- Zugang zu einem Netzwerk
- Akku-Betrieb oder andere mobile Stromquelle

Die VM für diese Konfiguration ist die KVM, die speziell dafür entwickelt wurde möglichst wenig Speicherplatz zu verwenden. Das „K“ von KVM steht für Kilo und soll zeigen wie wenig Speicher benötigt wird.

Um die VM möglichst kompakt zu bekommen wird der Funktionsumfang stark eingeschränkt. In der CLDC gibt es folgende Einschränkungen:

- **Keine Floating Point Unterstützung.** Die KVM unterstützt keine Kommazahlen, da die Berechnung solcher Zahlen sehr rechenintensiv sind und meist an spezielle Hardware gebunden ist, die in den meisten Geräten nicht vorhanden ist.
- **Kein Java Native Interface (JNI).** Es können keine Bibliotheken anderer Programmiersprachen verwendet werden. Der Hardwarehersteller kann in seine VM das JNI implementieren, dies ist aber nicht vorgeschrieben.
- **Keine Serialisierung von Objekten.**
- **Keine Thread-Gruppen oder Dameon-Threads.** Nur einfache Threads sind möglich.
- **Kein ClassLoader.** Die Anwendung kann nicht beeinflussen wie Klassen geladen werden. Nur die VM kann die Klassen laden.
- **Keine finalization() Methode oder schwache Referenzen.** Damit wird die Arbeit des Garbage-Collectors wesentlich einfacher und spart somit Ressourcen.
- **Klassen-Verifizierung wird nicht im J2ME-Programm vorgenommen**, sondern von dem Entwicklungsrechner (PC oder Server). Diese Aufgabe wird beim Erstellen der Anwendung vom sogenannten Preverifier übernommen.
- **Keine Reflections.**

In der CLDC werden teilweise Klassen aus J2SE übernommen. Diese sind in den Packages java.io, java.lang und java.util. Jedoch wurden die übernommenen Klassen wegen den oben genannten Einschränkungen teilweise stark eingeschränkt oder verändert.

Es gibt noch zusätzliche Klassen in der CLDC im sogenannten Generic Connection Framework (GCF). Dies ist eine Ansammlung von Klassen mit denen Verbindungen wie zum Beispiel Sockets oder HTTP-Requests ganz abstrakt abgebildet werden können. Dies entspricht bei J2SE einigen Klassen aus dem java.io- und java.net-Package. Das GCF implementiert dabei keine Protokolle. Die Protokolle werden entweder auf

Profile-Ebene oder im Endgerät definiert. Somit kann es passieren, dass einige Mobiltelefone einige Protokolle nicht unterstützen.

Tabelle 4: CLDC-Packages

Package	Beschreibung
java.io	Implementiert Klassen zur Ein- und Ausgabe durch DataStreams
java.lang	Eine Untermenge der J2SE-Klassen dieses Packages. Enthält die fundamentalen Klassen der Programmiersprache Java.
java.util	Innerhalb dieses Packages wird eine ganze Reihe von nützlichen Java-Klassen zusammengefasst. Hierzu zählen Klassen des Collection-Frameworks, der Datums- und Zeitbehandlung usw.
javax.mircoedition.io	Verbindungsklassen des GCF

Tabelle 5: J2SE 1.3 Klassen und Interfaces die in der CLDC eingebunden sind.

Package	Enthaltene Klassen und Interfaces
java.io	ByteArrayInputStream, ByteArrayOutputStream, DataInput, DataInputStream, DataOutput, DataOutputStream, InputStream, InputStreamReader, OutputStream, OutputStreamWriter, PrintStream, Reader, Writer
java.lang	Boolean, Byte, Character, Class, Integer, Long, Math, Object, Runnable, Runtime, String, StringBuffer, System, Thread, Throwable
java.util	Calendar, Date, Enumeration, Hashtable, Random, Stack, Time, Vector

Tabelle 6: Klassen und Interfaces des GCF die in der CLDC eingebunden sind.

Package	Enthaltene Klassen und Interfaces
javax.microedition.io	Connection, ConnectionNotFoundException, Connector, ContentConnection, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier

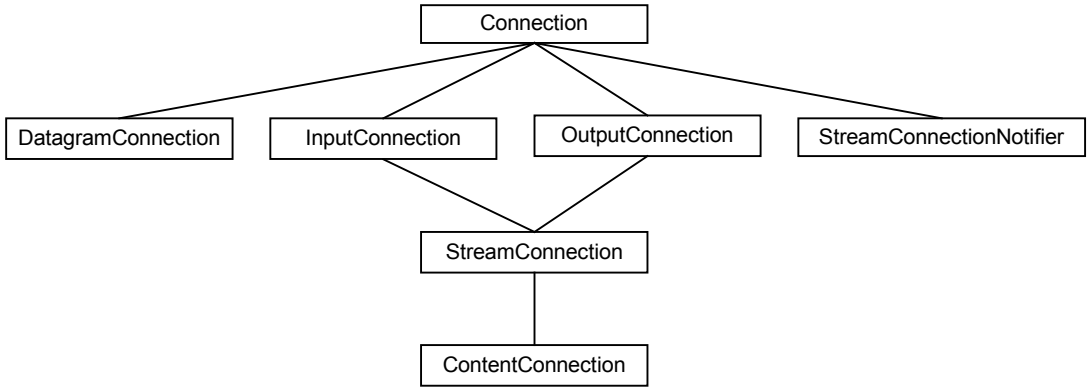

Abbildung 8: Klassenhierarchie im GCF

4.1.2. MIDP (Profil)

Das MIDP ist das Mobile Information Device Profile und baut auf die CLDC auf. Üblicherweise wird dieses Profil in Mobiltelefonen, kleinen PDAs und interaktiven Pagern verwendet. Das Profil setzt bestimmte Eigenschaften im Endgerät voraus:

- Schwarzweiß oder Farbdisplay mit mindestens 96 Pixel Breite und 54 Pixel Höhe mit einem Pixelseitenverhältnis von 1:1.

- Mindestens eines der folgenden Eingabemöglichkeiten: Telefontastatur, eine richtige Tastatur oder einen Touchscreen.
- Kabellosem 2-Wege Netzwerksupport.
- Mind. 128kB Speicher (Flash oder ROM) für die Anwendungen, mind. 8kB Speicher (Flash) für Daten der Anwendungen und mind. 32kB Speicher für den Java Laufzeitspeicher.
- Die KVM muss in einem eigenen Thread auf dem Betriebsystem des Endgeräts laufen.
- Persistant-Memory-Access für die Anwendungen, Zugang zu den kabellosen Netzwerkverbindungen, Schreibmöglichkeit auf dem Display und Zugriff auf die abgelaufene Zeit.
- Umwanden von Eingaben in Input-Events.
- Der MIDlet-Lifecycle (Installation, Auswahl, Starten, Schliessen und Löschen) muss verwaltet werden.

Die Verwaltung des MIDlet-Lifecycle übernimmt die Application Management Software auf dem Endgerät. Der genaue Vorgang wird jedoch nicht im MIDP definiert.

4.2. MIDlets

Eine J2ME-Applikationen nennt man MIDlet, wenn Sie auf das MIDP aufsetzen. Zum erstellen von MIDlets gibt es mehrere Möglichkeiten. Zum einen kann man KToolbar (welches im WirelessToolkit von SUN enthalten ist) für die Automatisierung der Abläufe verwenden, die einzelnen Schritte von Hand erledigen oder eine Entwicklungsumgebung für J2ME verwenden. Im praktischen Teil meiner Diplomarbeit habe ich das WirelessToolkit verwendet, auf welches ich später noch genauer eingehen werde.

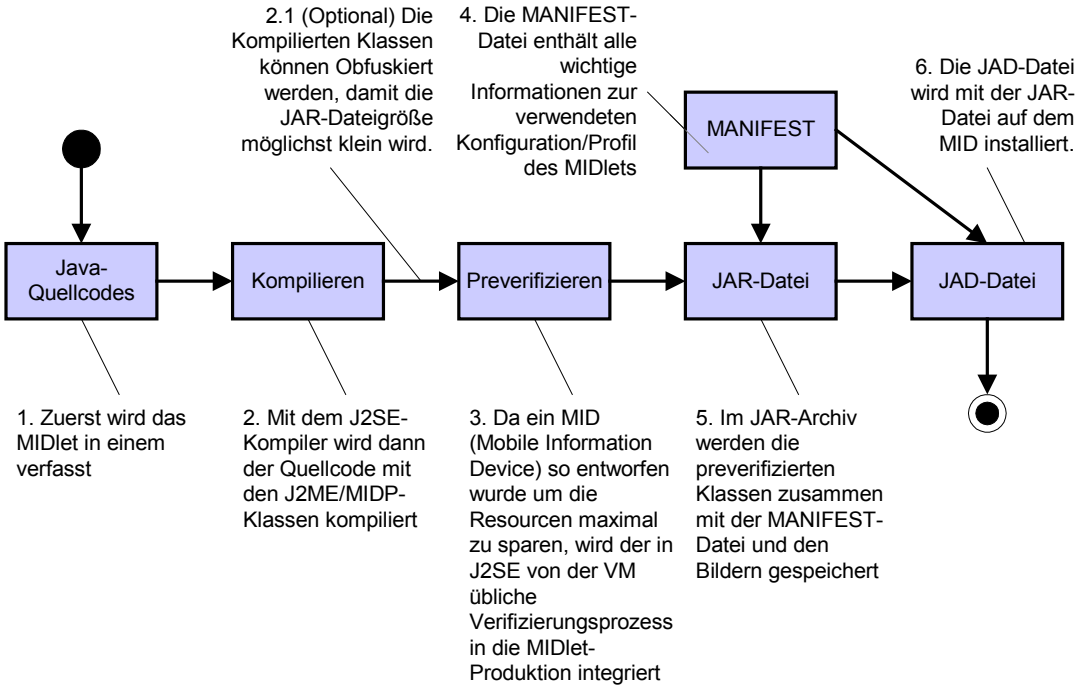

Abbildung 9: Schritte zum Erstellen eines MIDlets

Beim Erstellen des Quellcodes erbt die Hauptklasse des MIDlets von der Klasse javax.microedition.midlet.MIDlet (analog wie bei einem Applet), damit diese Applikation gestartet werden kann. Die Klasse MIDlet enthält die Methoden `startApp()`, `pauseApp()` und `destroyApp()`, die beim Ändern des MIDlet-Zustands vom System aufgerufen werden.

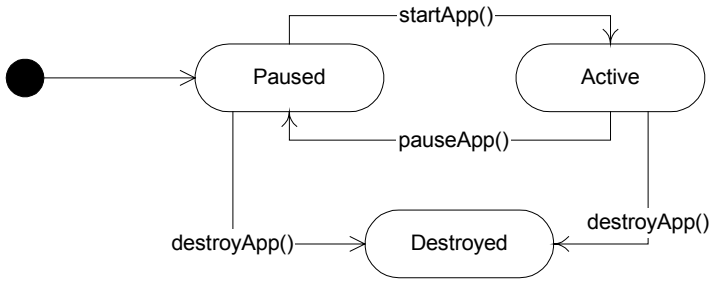

Abbildung 10: MIDlet-Zustände und die Änderungen

Die Java-Quellcodes werden mit dem J2SE-Compiler, der im J2SE-SDK enthalten ist, kompiliert. Dabei wird die im WTK enthaltene Datei midpapi.zip als Bootclasspath verwendet:

```
javac –bootclasspath \lib\midpapi.zip –d .\tmpclasses –classpath
.\tmpclasses .\src\*.java
```

So werden die Javaquelldateien im Verzeichnis src kompiliert und dann im Verzeichnis tmpclasses gespeichert.

Wer den durch das Kompilieren entstandenen Bytecode komprimieren will kann einen sogenannten Obfucator verwenden um damit alle Kommentare zu entfernen und lange Klassen- und Methodennamen zu verkürzen. Dadurch kann später die Grösse der fertigen JAR-Datei um gut 20% kleiner werden. Es gibt verschiedene kostenfreie Obfuscator-Programme im Internet. Retroguard ist einer von diesen, der auch mit dem WTK kompatibel ist. Den Obfuscator startet man wie folgt:

```
java retroguard .\tmpclasses\*.class
```

Nach diesem Schritt muss der Bytecode mit folgendem Befehl preverifiziert werden. Das notwendige Programm ist auch im WTK enthalten:

```
preverify –classpath \lib\midpapi.zip;.\tmpclasses –d .\classes
.\tmpclasses
```

Der preverifizierte Bytecode liegt nun im Verzeichnis classes.

Bevor wir jetzt die JAR-Datei erstellen benötigen wir die MANIFEST.MF-Datei, da diese mit in das JAR-Archiv gespeichert werden muss. Die MANIFEST.MF-Datei enthält Informationen über die verwendete Konfiguration (CLDC 1.0), das verwendete Profil (MIDP 1.0), die enthaltene MIDlets, den Namen und die Version der MIDlet-Suite und den Urheber.

```
MIDlet-1: mJabber, , com.mjabber.Mjabber
MIDlet-Name: mJabber
MIDlet-Vendor: mjabber.com
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Sobald wir die MANIFEST.MF-Datei haben können wir mit der Erstellung der JAR-Datei beginnen:

```
jar cmf MANIFEST.MF mJabber.jar –C .\classes .
```

Falls wir noch Bilder oder Icons in unsere MIDlet-Suite einbinden möchten können wir das mit der folgenden Anweisung machen:

```
jar umf MANIFEST.MF mJabber.jar –C .\res .
```

Zum Schluss benötigen wir noch die JAD-Datei. Diese enthält Informationen aus der MANIFEST.MF-Datei und noch die Grösse der JAR-Datei:

```
MIDlet-1: mJabber, , com.mjabber.Mjabber
MIDlet-Jar-Size: 38139
MIDlet-Jar-URL: mJabber.jar
MIDlet-Name: mJabber
MIDlet-Vendor: mjabber.com
MIDlet-Version: 1.0
```

Die JAR- und die JAD-Datei ergeben jetzt das sogenannte MIDlet-Suite. Diese kann entweder von einer WML-Seite aus OTA[11] oder über eine direkte Verbindung von einem PC auf das Endgerät geladen und installiert werden.

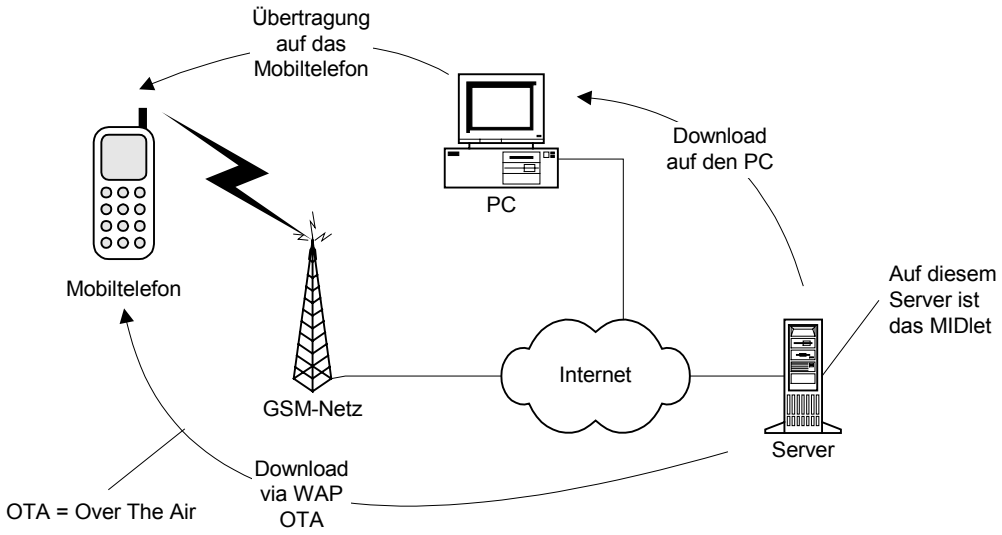

Abbildung 11: Laden eines MIDlets auf das Mobiltelefon

4.3. Die grafische Benutzerschnittstelle (GUI)

Für die Darstellung von Informationen, Bildern und Formularen auf dem Display wird das Package javax.microedition.lcdui benötigt. Alle darstellbaren Toplevel-Elemente implementieren das Interface Displayable. Diese Elemente nennt man Screens. Diese kann man dem Display des Endgeräts zuordnen.

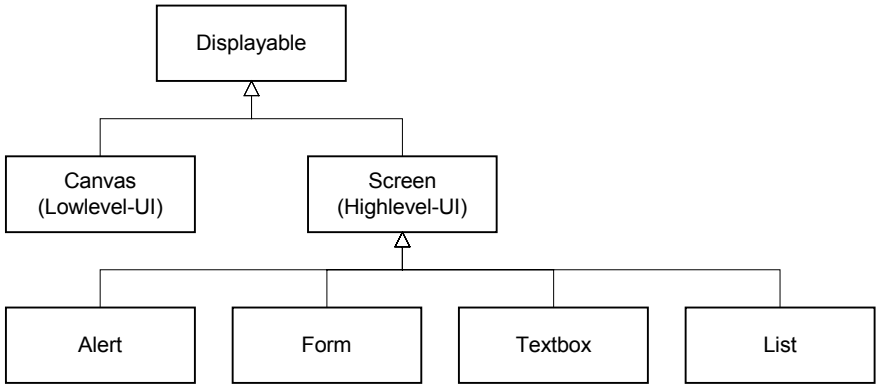

Abbildung 12: Die darstellbaren Toplevel-Elemente im MIDP

Das Lowlevel-Userinteface wird für Grafik, formatiertem Text und individuelle Tasteneingabe wie zum Beispiel bei Spielen verwendet. Das Highlevel-Userinterface ist für

11

Formulare, einfache Textdarstellung, Auswahllisten und Alarmnachrichten wie zum Beispiel bei einem IM-Client verwendet.

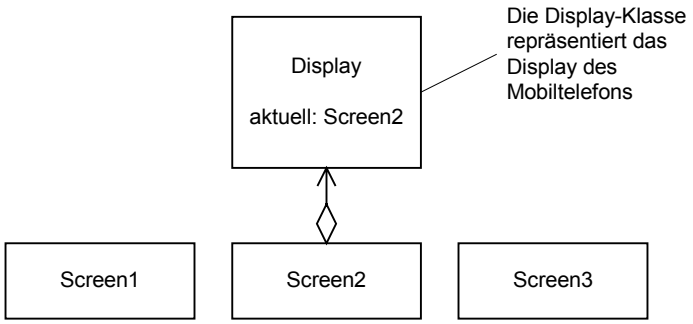

Abbildung 13: Zuordnung von Screens und Display

Um Tastatureingaben oder Menüpunkte auswählen zu können kann man einem Screen sogenannte Commands zuordnen. Ein Command entspricht der Button-Klasse in J2SE. Einen Command erstellt man zum Beispiel mit folgender Codezeile:

```
Command EXIT_CMD = new Command( "Text", Command.OK, 1 );
```

Command(String, int, int);	Konstruktor
"Text"	Der Namen des Commands, den der Benutzer später zu sehen bekommt
Command.OK	Bestimmt an welcher stelle der Command gesetzt werden soll. Command.OK ist die übliche Position eines OK-Buttons.
1	Beschreibt die Priorität. Wenn mehrere Commands dargestellt werden kann damit die Reihenfolge festgelegt werden.

Abbildung 14: Anlegen eines neuen Commands

Beim Ausführen eines Commands durch den Benutzer wird die Methode `commandAction()` im CommandListener mit dem aktuellen Screen und dem ausgeführten Command aufgerufen.

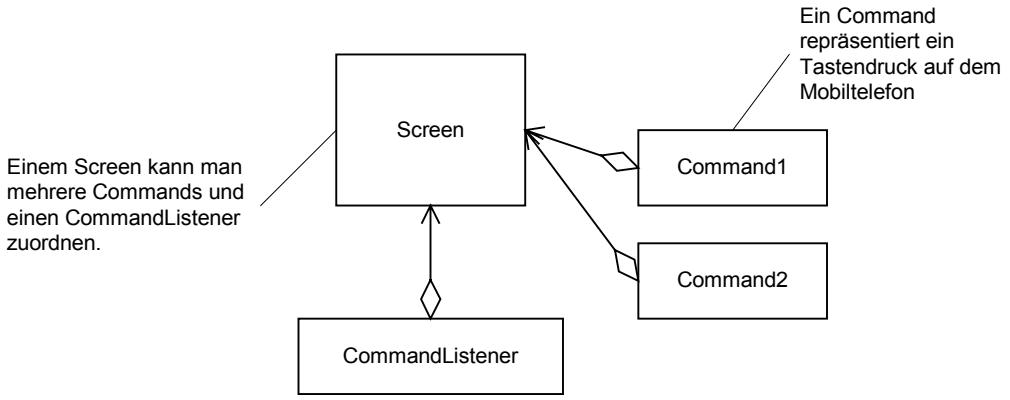

Abbildung 15: Zuordnung von Commands, CommandListener und Screens

4.4. Record Store Management (RMS)

J2ME verfügt neben dem Speicher für die MIDlets und dem Arbeitsspeicher (Heap-Memory) auch über einen persistenten Speicherbereich (Persistant-Storage), indem Daten nach Programmende weiter existieren. Auf diesen Speicherbereich kann man über das RMS zugreifen. Für die Implementierung des RMS ist jeder Hardwarehersteller selbst verantwortlich. Das ist der Grund, warum die Einschränkungen von Gerät zu Gerät unterschiedlich sind. Bei einem Palm darf zum Beispiel die Objektgröße des gespeicherten Records nicht größer als 64 kB sein, da das PalmOS keine größeren Records verwalten kann.

Von der J2ME-Seite aus können beliebig viele Records angelegt werden. Es können auch mehrere Records pro MIDlet verwendet werden. Jeder Record benötigt allerdings einen eindeutigen Namen innerhalb einer MIDlet-Suite. Eine MIDlet-Suite ist ein JAR-Archiv indem mehrere MIDlets enthalten sind. Was MIDlets sind und wie man welche erstellen kann wird noch später genauer beschrieben.

Anhand dieses Codebeispiels kann man sehen wie man einen Recordstore öffnen kann, der dann die verschiedenen Records enthält:

```
RecordStore rs = null;
try {
   rs = RecordStore.openRecordStore("RecordStoreDatei", true);
}
catch (RecordStoreNotFoundException e) {}
catch (RecordStoreFullException e) {}
catch (RecordStoreException e) {}
```

Um jetzt auf ein Record zugreifen zu können muss man wissen an welcher Stelle der Record stand. In meinem Beispiel ist er an der Stelle 1. Der Rückgabewert von `rs.getRecord(int)` ist ein Byte-Array:

```
String name = new String( rs.getRecord( 1 ) );
```

Auf die gleiche Weise kann man die Daten in ein Records wieder hinein schreiben:

```
String name = "Hans"
byte[] name_byte = name.getBytes();
rs.addRecord( name_byte, 0, name_byte.length );    // für neue Records
rs.setRecord( 1, name_byte, 0, name_byte.length ); // zum Überschreiben
```

Mit der Methode `addRecord()` wird ein neuer Record hinzugefügt. Mit der Methode `setRecords()` wird ein bestehender Record überschrieben.

4.5. Das Wireless Toolkit am Beispiel des HelloWorld-MIDlets

SUN bietet mit dem WTK (Wireless Toolkit) für J2ME-Entwickler eine Möglichkeit die vielen Arbeitsschritte beim Erstellen von MIDlets zu automatisieren. Zudem ist im WTK ein Emulator enthalten, mit dem die MIDlets auf verschiedenen Endgeräten, wie zum Beispiel einem Palm PDA oder einem Mobiltelefon, (emuliert) getestet werden können.

Fangen wir zunächst mit dem Quellcode für das HelloWorld-MIDlet an. Dieser sieht wie folgt aus:

```
// Import der notwendigen Packages
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {

    // Screen und Command-Objekt
    private Form myScreen;
    private Command EXIT_CMD = new Command( "Exit", Command.EXIT, 1 );

    // Methode, die beim Start des MIDlets aufgerufen wird
    protected void startApp() {
        myScreen = new Form( "MIDlet-Titel" );
        myScreen.addCommand( EXIT_CMD );
        myScreen.setCommandListener( this );
        myScreen.append( "Hello World!" );
        Display.getDisplay( this ).setCurrent( myScreen );
    }

    // Methode, die beim wechseln in den Pause-Zustand aufgerufen wird
    protected void pauseApp() {}

    // Methode, die beim Beenden aufgerufen wird
    protected void destroyApp(boolean unconditional) {}

    // Methode des CommandListener-Interface
    public void commandAction( Command c, Displayable d ) {
        if ( c == EXIT_CMD ) {
            destroyApp( true );
            notifyDestroyed();
        }
    }

}
```

Die Methoden `startApp()`, `pauseApp()` und `destroyApp()` werden von der Klassen MIDlet geerbt und müssen implementiert werden. Beim Start des Applets wird die Methode `startApp()` aufgerufen. Diese Methode erzeugt das Screen-Objekt `myScreen` und fügt diesem einen Text und den `EXIT_CMD` zu. Danach wird mit der statischen Methode `Display.getDisplay()` das Display-Objekt des MIDlets aufgerufen. Das Screen-Objekt wird dann mit der Methode `setCurrent()` als aktuellen Screen gesetzt und somit angezeigt.

Die Methode `commandAction()` wird über das Interface CommandListener implementiert. Diese Methode wird aufgerufen, wenn zum Beispiel ein Button gedrückt wird.

Dabei wird der Methode der aktuelle Screen und der gedrückte Button übermittelt. Das Command-Objekt `EXIT_CMD` ist der Button, der zum Beenden des MIDlets benötigt wird. Dieser wird dem Form (Subklasse von Screen) zugewiesen.

Wenn man KToolbar startet sieht man folgendes Fenster. In diesem Fenster wählt man „New Project“ um ein neues Projekt anzulegen. Wir legen das Projekt „Hello World“ an.

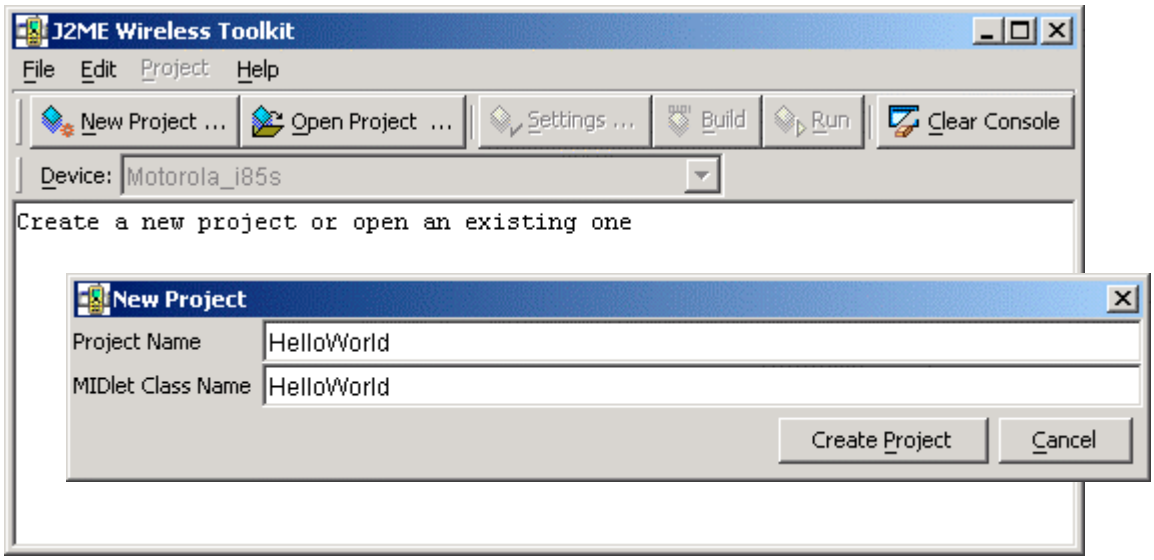

Abbildung 16: KToolbar aus dem WirelessToolkit von SUN

Durch das Anlegen des neuen Projekts wurde im Verzeichnis „C:\WTK104\apps\“ ein neues Unterverzeichnis mit dem Projektnamen erzeugt. In diesem Projektverzeichnis sind wiederum vier Unterverzeichnisse mit den Namen „bin“, „lib“, „res“ und „src“.

Tabelle 7: Verzeichnisse bei einem WTK-Projekt

Verzeichnis	**Beschreibung**
bin	Hier speichert KToolbar die MANIFEST.MF-, JAD- und JAR-Dateien
lib	In diesem Verzeichnis können zusätzliche Libraries kopiert werden
res	Alle Bilder und Icons sollen in diesem Verzeichnis gespeichert werden, damit KToolbar diese dann in das JAR-Archiv mit aufnehmen kann.
src	In diesem Verzeichnis liegen die Quellcodes
classes	Hier speichert KToolbar preverifizierten Klassen ab
tmpclasses	KToolbar speichert die kompilierten, aber noch nicht preverifizierten, Klassen in diesem Verzeichnis
tmplib	Diese Verzeichnis wird auch zum Kompilieren der Libraries benötigt

Dann erscheint ein neues Fenster bei dem die verschiedenen Informationen über das MIDlet eingegeben werden können. Diese Daten werden dann in der Datei „MANIFEST.MF“ und in „HelloWorld.jad“ im „bin“-Verzeichnis gespeichert. Unsere Java-Quellcode-Datei speichern wir im „src“-Verzeichnis („src“ steht für Source = Quelle).

Nachdem wir das gemacht haben können wir mit „Build“ unseren Quellcode kompilieren. Dabei wird automatisch der kompilierte Quellcode preverifiziert und in ein HelloWorld.jar-Archiv gepackt. Danach können wir mit „Run“ unser MIDlet ausführen und

testen. Es können dabei verschiedene Endgeräte emuliert werden. Das sieht dann ungefähr so aus wie in der nächsten Abbildung.

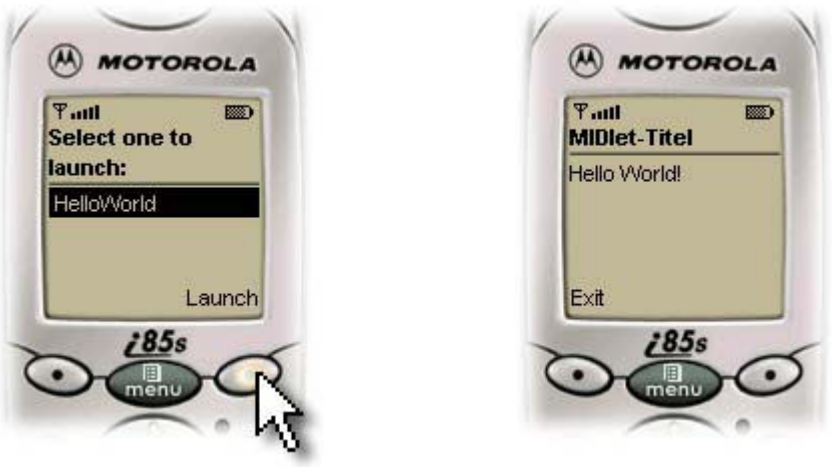

Abbildung 17: Ausschnitte aus dem WTK- Emulator-Fenster

4.6. Weitere APIs

In Zukunft werden weitere APIs von den Hardwareherstellern in ihren Endgeräten implementiert werden. Dazu hat SUN unter anderem folgende APIs im sogenannten JSRs (Java Specification Request) vorgestellt:

- **Mobile Media API** für die Wiedergabe von Video, Ton, und sonstigen Medien
- **Wireless Messaging API**
- **Bluetooth API** zur Kommunikation mit anderen Bluetooth-Geräten
- **Java Speech API**
- **Mobile Game API** zur einheitlichen Unterstützung von spieletypischen Geräteeigenschaften wie Vibrator, Sound und Grafik
- **Mobile 3D Graphics API** für die Darstellung von 3D-Objekten
- **Location API** für Location Based Services wie zum Beispiel ein Routenplaner

4.7. Typische Probleme und deren Lösungsansätze

Bei der Entwicklung von Java-Anwendungen für J2ME stößt man immer wieder auf Probleme, die durch die Beschaffenheit der Geräte entstehen. Dazu zählt die fehlende Fließkommazahlen-Unterstützung, der geringe verfügbare Speicherplatz, die langsamen Übertragungsgeschwindigkeiten, die langsame CPU, nicht vorhandene Verschlüsselungsverfahren und die Serialisierung von Objekten.

4.7.1. Keine Fließkommazahlen

Eines der Probleme der CLDC sind die nicht implementieren Fließkommazahlen. Möchte man zum Beispiel eine räumliche Darstellung eines Drahtgittermodels darstellen und dieses dann um einen bestimmten Winkel drehen, so benötigt man für die Berechnung der Koordinaten die Sinus-Funktion. Aber wie soll man den Sinus-Wert ohne Fließkommazahlen berechnen?

Um das Problem der Sinus-Funktion genauer zu betrachten stellen wir uns dazu die mathematische Formel y = sin x vor. Man kann weder den x-Wert, der bekanntlich zwischen 0 und 2π liegen soll, als Fließkommazahlen übergeben, noch das Ergebnis, welches nur zwischen -1 und 1 liegen darf, wieder korrekt speichern. Ohne Fließkommazahlen könnte y nur die Werte -1, 0 und 1 einnehmen und x die Werte 0, 1, 2, 3, 4, 5 und 6. Damit wäre die Sinus-Funktion so ungenau, dass diese für keine Rechnung zu gebrauchen ist.

Es gibt grundsätzlich zwei Möglichkeiten mit Ganzen Zahlen eine Fließkommazahlen darzustellen. Eine Möglichkeit ist die Zahl vor dem Komma als eine Ganze Zahl und die Zahl hinter dem Komma als andere Ganze Zahl zu definieren.

```
float f = 3.014;
```

wird dann zu

```
int vorkomma = 3;
int nachkomma = 14;
int kommastellen = 3;
```

Bei der anderen Möglichkeit wird einfach die Fließkommazahl mit einem Faktor 10^n multipliziert um die Stellen hinter dem Komma vor das Komma zu verschieben. n gibt in diesem Fall die Anzahl der Stellen an, die nach dem Komma berücksichtigt werden.

```
float f = 3.014;
```

wird dann zu

```
int zahl = 3014;                // weil zahl = f * 10^n
int kommastellen = 3;           // und kommastellen = n
```

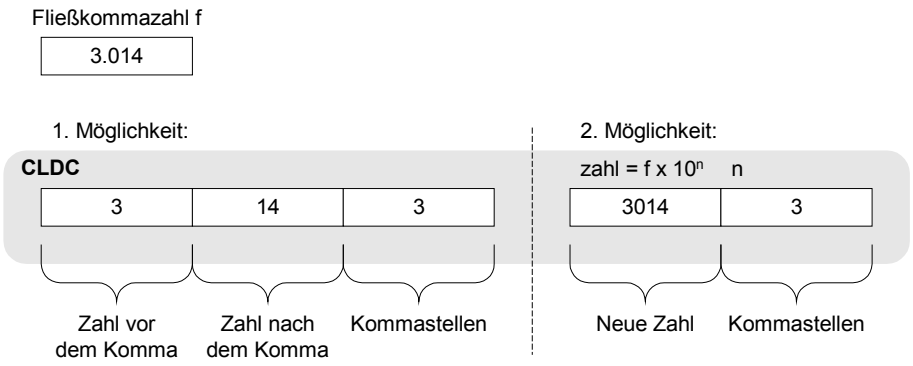

Abbildung 18: Mögliche Darstellung von Fließkommazahlen in der CLDC

4.7.2. Wenig Speicher

Da in den mobilen Endgeräten wie Mobiltelefonen der Speicherplatz nur in geringem Maße zur Verfügung steht, muss man versuchen so effizient wie möglich mit dem vorhandenen Speicher umzugehen.[12]

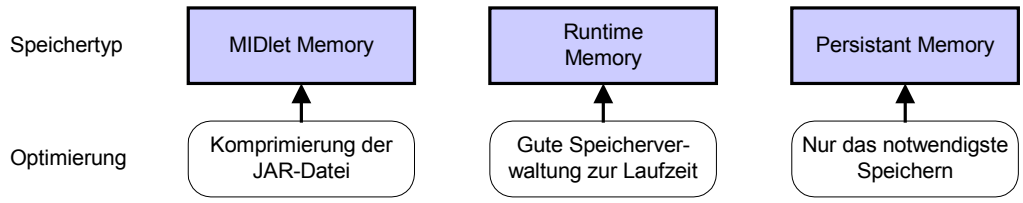

Abbildung 19: Speicheroptimierung

4.7.2.1. MIDlet-Memory

Da der Speicherplatz für MIDlets in manchen Mobiltelefonen sehr begrenzt ist (zum Teil sogar kleiner 30KB) sollte man nur die notwendigsten Funktionen ist das MIDlet implementieren. Man spricht dabei von Thin-Clients. Alle unnötigen Funktionen machen das MIDlet unnötig groß. Wenn man den Funktionsumfang schon auf das notwendigste beschränkt hat gibt es aber dennoch Tricks die Größe der JAR-Datei, welche die MIDlets enthält, möglichst klein zu bekommen.

Im Java-Bytecode werden komplette Methodennamen, Objektnamen und Kommentare mit gespeichert. Deshalb kann man durch die Entfernung von Kommentaren, und die Umbenennung von Methoden- und Objektnamen viel Speicherplatz einsparen. Sogenannte Obfuscator-Programme machen genau dies. Die Obfuscator wurden eigentlich dazu entwickelt den Java-Bytecode dahingehend zu verändern, damit durch Dekompilieren kein verständlicher Quellcode entsteht. Hier ein kleines Beispiel:

```
public class Testklasse {
   private String name;
   public String getName() {
      return name;
   }
   public void setName( String neuername ) {
      name = neuername;
   }
}
```

Der Obfuscator würde aus diesem Javacode folgenden Code produzieren:

```
public class a {
   private String a;
   public String a() { return a; }
   public void b( String b ) { a = b; }
}
```

[12] [J2ME]: S. 54 – 62

Durch dieses Verfahren wird die Größe der JAR-Datei um bis zu 20% kleiner. Der Obfuscating-Vorgang muss bei J2ME-Programmen vor dem Preverify-Vorgang stattfinden, da sonst alle Daten die der Preverify-Vorgangs in den Klassen gespeichert hat wieder rückgängig gemacht werden.

4.7.2.2. Runtime-Memory

Der Runtime-Memory ist der Arbeitsspeicher, der während dem Ausführen des Programms zur Verfügung steht. Dieser besteht aus dem Stack und dem Heap. Bei Java werden alle Objekte immer im Heap abgelegt, einfache Datentypen (wie zum Beispiel `int`, `double` oder `boolean`) und die Referenz zu den Objekten werden im Stack abgelegt. Es gibt folgende Tricks um den Arbeitsspeicher möglichst gering zu belasten:

- Einfache Variablentypen statt Objekte verwenden
- Unbenutzte Objekte für den Garbage-Collector vorbereiten
- Exceptions und anderes unnötiges Erzeugen von Objekten vermeiden

Bei der Instanzierung von Objekten werden diese zur Laufzeit im Heap-Memory angelegt. Dabei wird zum einen mehr Speicherplatz benötigt als bei einfachen Variablen und zum anderen belastet das Anlegen von Objekten die CPU deutlich stärker.

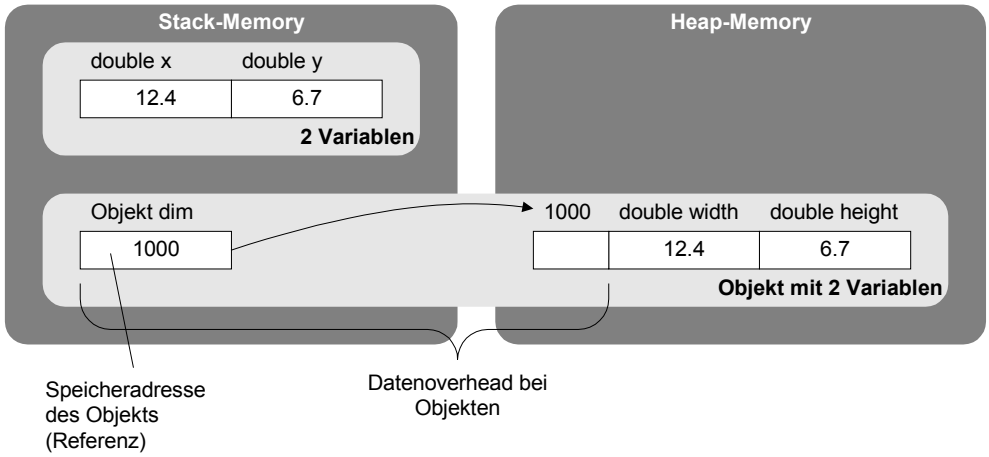

Abbildung 20: Unterschiedlicher Speicherverbrauch zwischen Objekten und einfachen Variablentypen

Der Garbage-Collector ist dazu da, Objekte, die nicht mehr benötigt werden, zu löschen um den Speicherplatz für neue Objekte wieder frei zu machen. Dieser Vorgang ist meist sehr komplex und rechenintensiv. Deshalb sollte man alle Referenzen von Objekte, die nicht mehr benötigt werden, auf `null` setzen. So findet der Garbage-Collector leichter die unbenutzten Objekte und kann diese schneller löschen.

Die dritte Möglichkeit den Heap-Speicher effizienter zu nutzen ist das Vermeiden von Exceptions. Bei einem Fehler im Java-Programm wird ein Exception-Objekt erzeugt welches dann von try/catch-Anweisungen verarbeitet werden kann. Durch das Erstellen des Exception-Objekts wird zum einen meist unnötig Speicher belegt und dieser

Vorgang belastet zudem die Ressourcen der CPU. Besser ist es mögliche Exceptions mit if-Anweisungen abzufangen, bevor diese überhaupt erzeugt werden.

4.7.3. Kleine Bandbreite

Bei den meisten mobilen Endgeräten steht nur Übertragungstechniken mit einer relativ langsamen Geschwindigkeit zur Verfügung. Daher ist es notwendig den Datenverkehr möglichst klein zu halten. Bei der Übertragung von einheitlich strukturierten XML-Daten gibt es zum Beispiel ein Komprimierungsverfahren, welches unter dem Namen WBXML[13] bekannt ist. Dabei wird ein XML-Dokument als Daten-Stream angesehen und jedes XML-Tag als Token. Dieses Verfahren muss allerdings von beiden Seiten (Sender und Empfänger) unterstützt werden. Auf die eine genauere Erklärung dieses Verfahrens verzichte ich, da dies den Rahmen dieser Arbeit sprengen würde. Ein Beispiel ist im Internet unter http://wireless.java.sun.com/midp/ttips/compressxml zu finden.

4.7.4. Langsame CPU

Die CPU der meisten mobilen Endgeräte ist sehr langsam und daher muss hier speziell auf die Verwendung der zur Verfügung stehenden Ressourcen geachtet werden.[14] Dazu gibt es verschiedene Dinge die beachtet werden sollten:

- **Objekte wiederverwenden**, da das Anlegen und Löschen eines Objekts im Heap ist sehr rechenintensiv ist. Es ist meist schneller einem bestehenden Objekt neu Werte zuweisen, statt ein völlig neues Objekt zu erzeugen.
- **Lokale Variablen verwenden**, da der Zugriff auf Klassen- oder Objektvariablen langsamer ist. Bei Schleifen, die immer wieder auf die gleiche Variable zugreifen ist dies besonders sinnvoll.
- **Verkettungen von String-Objekten vermeiden**, da bei einer Verkettung automatisch ein neues StringBuffer-Objekt erzeugt wird, welches dann wieder mit der toString-Methode in einen neues String-Objekt umgewandelt wird.
- **Exceptions vermeiden**, da bei einer Exception automatisch das entsprechende Exception-Objekt erzeugt wird und das Erzeugen von Objekten sehr rechenintensiv ist.
- **Garbage-Collector zur richtigen Zeit einsetzen**. Da dieser Vorgang sehr rechenintensiv ist, sollte man den Garbage-Collector manuell starten, wenn der Benutzer des Programms in diesem Zeitpunkt gerade keine rechenintensive Funktion ausführt (zum Beispiel während des Darstellens eines Formulars).

---

[13] [WBXML]

[14] [J2ME] S. 62 – 65

- **Bitweise shiften statt multiplizieren oder dividieren**. Dieser Vorgang ist deutlich schneller. Möchte man eine Zahl mit 2 multiplizieren, so kann man die Bits dieser Zahl auch um ein Bit nach links verschieben. Bei der Division durch 2 kann man auch die Bits der Zahl um ein Bit nach rechts verschieben. Für die Multiplikation `y = x * f` kann man `y = x << n` und für die Division `y = x / f` kann man `y = x >> n` schreiben, wobei folgendes gilt: $f = 2^n$ mit $n \in N$

4.7.5. Kleine Displays, beschränkte Eingabemöglichkeiten

Da gerade bei Mobiltelefonen meist nur ein sehr kleines Display und eine kleine Tastatur (12-Tasten) zur Eingabe zur Verfügung steht muss hier besonders auf eine logische, gut durchdachte und benutzerfreundliche Benutzerführung und Darstellung geachtet werden. Ein perfekte Lösung für alle Programme gibt es nicht, da in jedem Fall neu überlegt werden muss was logisch und benutzerfreundlich ist. Man sollte bei diesen Überlegungen einfach darauf achten, dass der Benutzer möglichst wenig Interaktion für die wichtigsten Funktionen des Programms benötigen sollte.

4.7.6. Verschlüsselungsverfahren

In der CLDC und im MIDP sind keine Verschlüsselungsverfahren enthalten. Wenn man aber trotzdem mit einem MIDlet Informationen verschlüsseln will (sei es für die sichere Übertragung in ein Netzwerk oder zum sicheren Speichern mit dem RMS), kann man entweder die sehr komplexen Verschlüsselungsalgorithmen selbst implementieren oder viel besser auf eine Klassenbibliothek des OpenSource-Projekts „The Legion of the Bouncy Castle“[15] zugreifen. Diese unterstützen eine Vielzahl von verschiedenen Verschlüsselungsalgorithmen wie zum Beispiel DES, IDEA, RC4 oder RSA und Hashfunktionen wie MD5 oder SHA1. Dies ist nur ein kurzer Auszug aus dem Funktionsumfang dieser Klassenbibliothek. Auf die Verwendung dieser Bibliothek wird aufgrund der großen Komplexität des Themas „Verschlüsselungsverfahren“ nicht weiter eingegangen.

4.7.7. Serialisierung von Objekten

Die Serialisierung von Objekten ist in der CLDC nicht enthalten. Jedoch besteht die Möglichkeit eine eigene Serialisierung bei entsprechenden Objekten zu implementieren.[16] Dazu müssen zwei Methoden in diese Klasse implementiert werden. Eine Methode erzeugt aus den Objektvariablen ein ByteArray, welches dann im Persistant-Memory gespeichert werden kann (im Beispiel heißt diese Methode `getData()`). Die andere Methode (im Beispiel `setData()` genannt) wandelt ein ByteArray wieder in die Werte der Objektvariablen um. Hier ein kleines Beispiel:

[15] http://www.bouncycastle.org
[16] [SERIALIZATION]

```
public Testklasse {

   public String name;
   public int alter;

   public byte[] getData() {
      ByteArrayOutputStream bout = new ByteArrayOutputStream();
      DataOutputStream dout = new DataOutputStream( bout );
      dout.writeUTF( name );
      dout.writeInt( alter );
      dout.flush();
      return bout.toByteArray();
   }

   public void setData( byte[] data ) {
      ByteArrayInputStream bin = new ByteArrayInputStream( data );
      DataInputStream din = new DataInputStream( bin );
      name = din.readUTF();
      alter = din.readInt();
   }
}
```

5. Der mobile Instant-Messaging Prototyp

Zuerst wird erläutert was man unter dem Begriff Instant-Messaging (IM) versteht. Instant-Messages (IM's) sind Kurznachrichten, die ähnlich wie Emails, über das Internet verschickt werden. Wenn man Emails mit richtigen Briefen vergleicht, dann entspricht eine IM einem Zettel mit einer kleinen Nachricht. Der Erfolg von IM liegt meiner Meinung nach in zwei Faktoren: Zum Ersten unterstützen alle IM-Systeme einen Dienst bei dem der Benutzer sehen kann welcher seiner Freunde gerade auch mit dem System verbunden und somit online ist und zum Zweiten sind IM wesentlich schneller geschrieben, weil diese formlos sind und somit eine förmliche Anrede, der Betreff, usw. nicht geschrieben werden muss. Eine typische IM könnte lauten: „Hallo, was machen wir heute Abend?" Die momentan bekanntesten IM-Systeme heißen ICQ, AIM, MSM, Yahoo!-Messenger oder Jabber[17]. Alle diese Systeme können mit einem kostenlosen Client frei verwendet werden. Jabber hat als einziges System ein offenes XML-basiertes Protokoll; alle anderen Systeme setzen auf ein eigenes properitäres Protokoll.

5.1. Szenario

Und wie funktioniert eigentlich ein IM-Client? Diese Frage ist ganz einfach zu beantworten: Alle bedeutenden IM-Systeme sind, was das Versenden von IM's betrifft, als Client-Server-Systeme realisiert. Dies bedeutet, dass alle Clients nur mit dem Server kommunizieren. Wenn also ein Benutzer von einem PC an einen anderen PC eine Nachricht schicken will, wird diese von seinem Client an den zuständigen Server verschickt. Dieser schickt dann diese Nachricht an den Empfänger weiter. Falls der Empfänger nicht mit dem Server verbunden ist, wird die Nachricht solange auf dem Server zwischengespeichert, bis die Nachricht zugestellt werden kann.

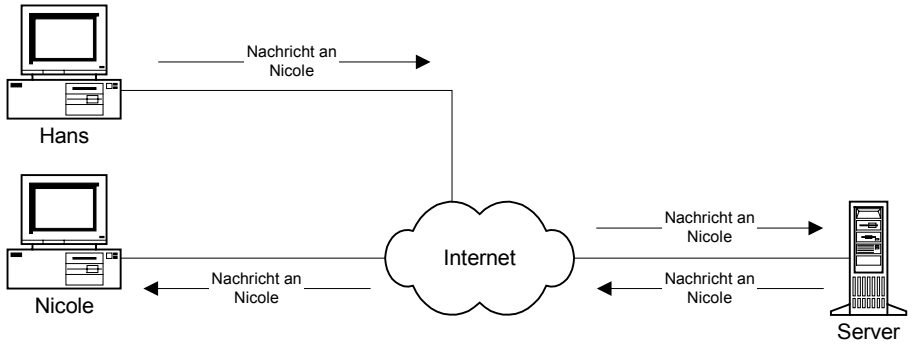

Abbildung 21: Client-Server-Architektur bei IM-Systemen

[17] [JABBER]

Die meisten IM-Systeme bieten noch einige weitere Dienste, wie zum Beispiel File-Transfer oder Internet-Telefonie mit VoIP an. Auf diese Dienste wird in dieser Arbeit nicht weiter eingegangen, da diese momentan für Mobiltelefone keine Rolle spielen.

Im Fall des mobilen IM-Clients unterscheidet sich das Szenario nur etwas von dem oben genannten: Mindestens ein Client läuft auf einem Mobiltelefon, welches über den Mobilfunkprovider mit dem Internet verbunden ist.

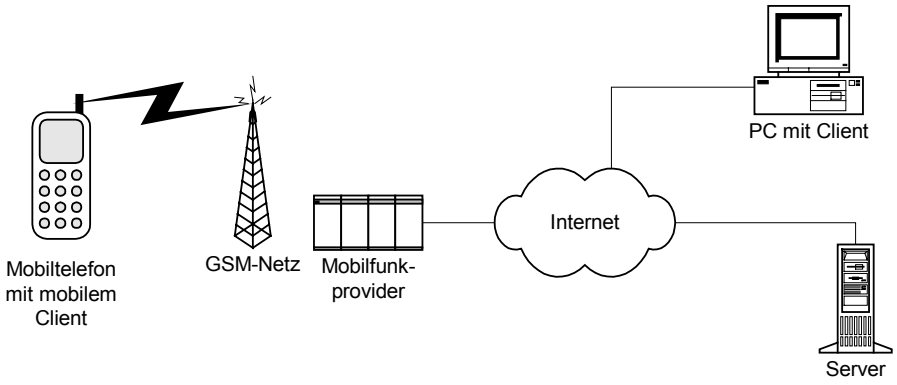

Abbildung 22: Instant-Messaging-System mit mobilem Client

5.2. Anforderungsanalyse

Um einen mobilen IM-Client entwickeln zu können, muss zuerst festlegen werden, welche Funktionen in diesem Client implementiert werden sollen. Da in den Mobiltelefonen nur sehr wenig Speicher verfügbar ist beschränkt sich der Prototyp, nur auf die notwendigsten Funktionen des IM.

Precondition für alle Use-Cases

- Programm muss auf dem Mobiltelefon installiert sein.
- Eine Verbindung zum Internet muss bestehen (zum Beispiel via GPRS)

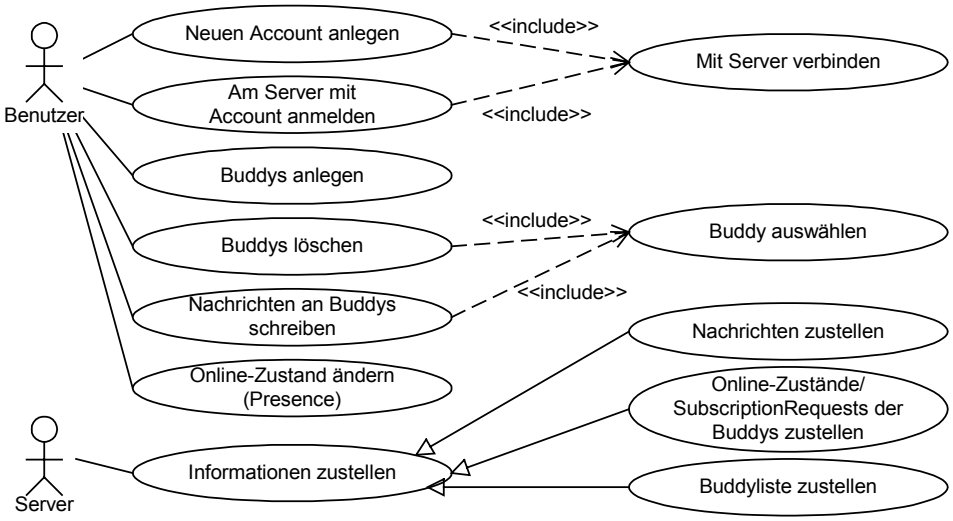

Abbildung 23: Use-Case-Diagramm

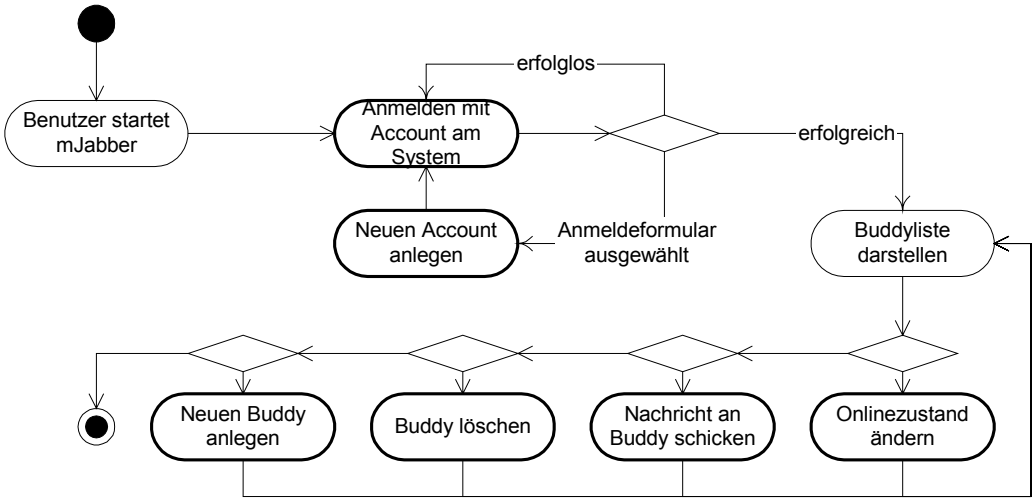

Abbildung 24: Verknüpfung der einzelnen Benutzer-Use-Cases

5.2.1. Use-Case: Neuen Account anlegen

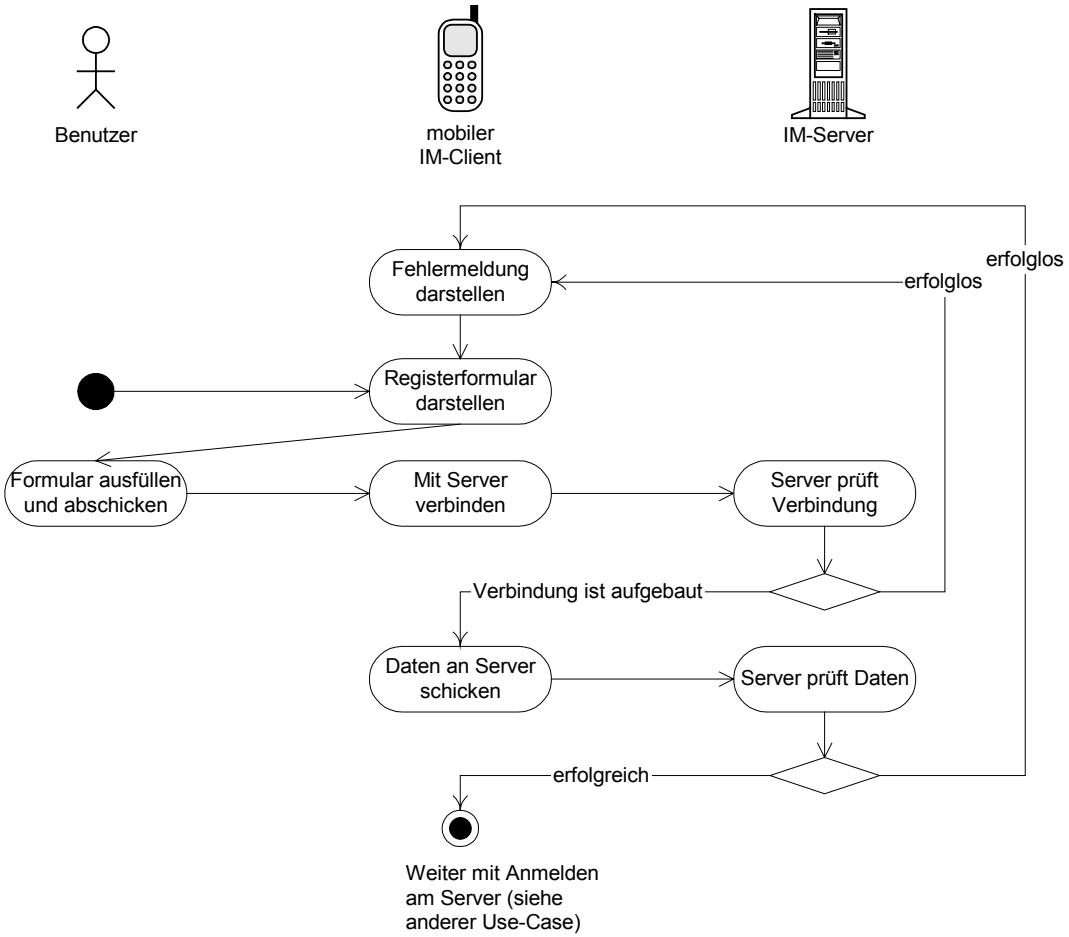

Abbildung 25: Activity-Diagramm des Use-Case Anlegen eines neuen Accounts

Basic-Flow:

1. Dieser Use-Case startet mit dem Auswählen des Menüpunktes „Neuen Account anlegen“ im Login-Screen.
2. Der Benutzer füllt das Anmeldeformular mit Name, Benutzerkennung, Passwort, Server und Ressource aus und schickt das Formular ab.
3. Der mobile IM-Client baut eine Verbindung zum Server auf und schickt die Anmeldedaten an den Server.
4. Der Server prüft ob eine Anmeldung mit den Daten funktioniert und schickt eine Anwort an den mobilen IM-Client zurück.
5. War das Anlegen des Accounts erfolgreich so wird eine Anmeldung mit den Accountdaten am Server gestartet (siehe nächster Use-Case).

Alternate-Flows:

- Falls keine Verbindung zum Server aufgebaut werden kann wird eine Fehlermeldung dargestellt und danach das Anmeldeformular wieder angezeigt.
- Falls die Anmeldedaten nicht korrekt sind, wird eine Fehlermeldung dargestellt und danach das Anmeldeformular wieder angezeigt.

Preconditions:

- Keine

Postconditions:

- War das Anlegen des Accounts erfolgreich so wird eine Anmeldung mit den Accountdaten am Server gestartet (siehe nächster Use-Case)

5.2.2. Use-Case: Am Server mit Account anmelden

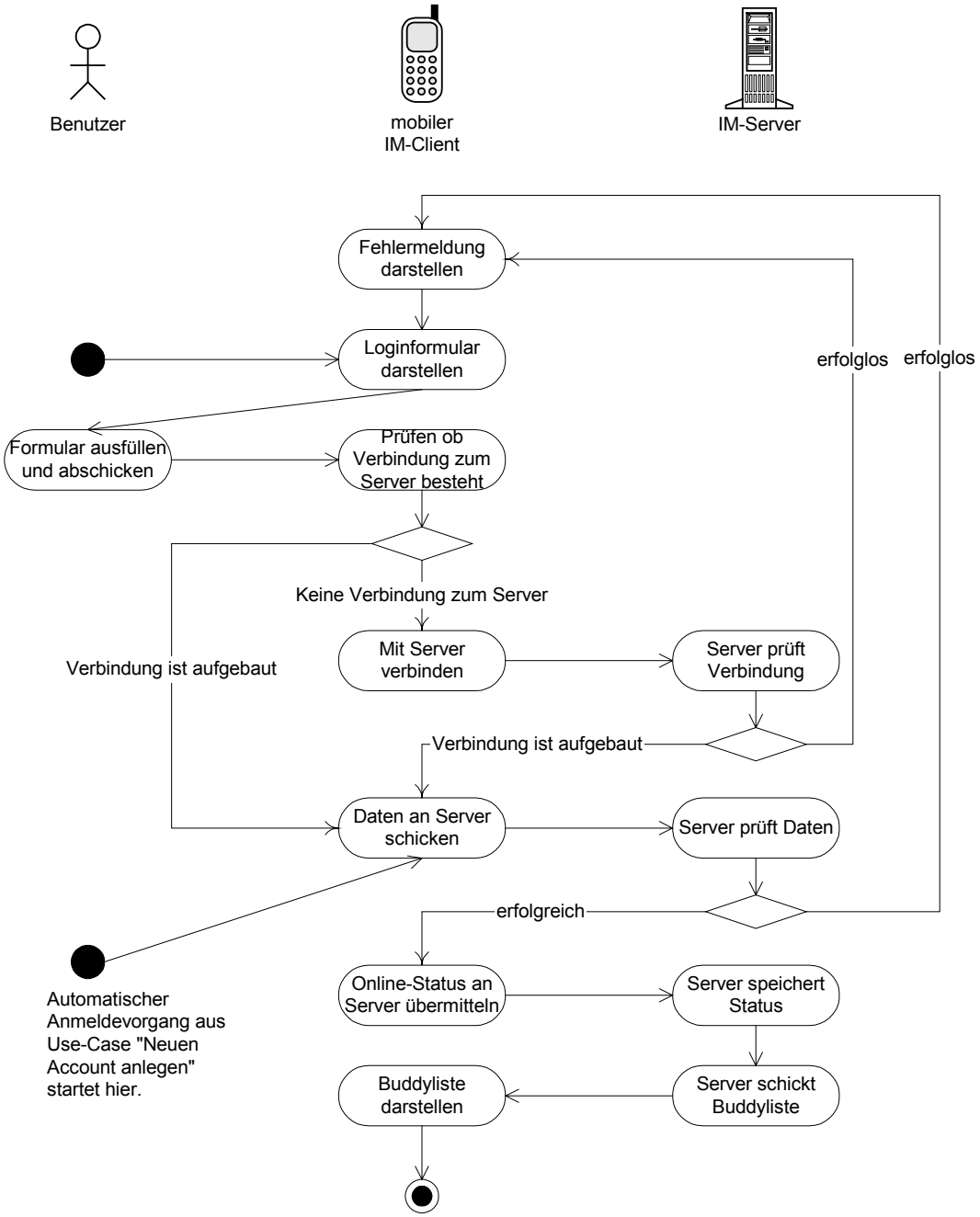

Abbildung 26: Activity-Diagramm des Use-Case Am Server mit Account anmelden

Basic-Flow

1. Dieser Use-Case startet mit dem Darstellen des Login-Screens.
2. Der Benutzer füllt das Loginformular mit Benutzerkennung, Passwort, Server und Ressource aus und schickt das Formular ab.
3. Der mobile IM-Client baut eine Verbindung zum Server auf und schickt die Anmeldedaten an den Server.

4. Der Server prüft ob eine Anmeldung mit den Daten am System funktioniert und schickt eine Anwort an den mobilen IM-Client zurück.
5. War die Anmeldung am Server mit den Accountdaten erfolgreich so wird die Buddyliste vom Server dargestellt.

Alternate-Flows:

- Falls keine Verbindung zum Server aufgebaut werden kann wird eine Fehlermeldung dargestellt und danach das Anmeldeformular wieder angezeigt.
- Falls die Anmeldedaten nicht korrekt sind, wird eine Fehlermeldung dargestellt und danach das Anmeldeformular wieder angezeigt.

Preconditions:

- Keine

Postconditions:

- Keine

5.2.3. Use-Case: Neuen Buddy anlegen

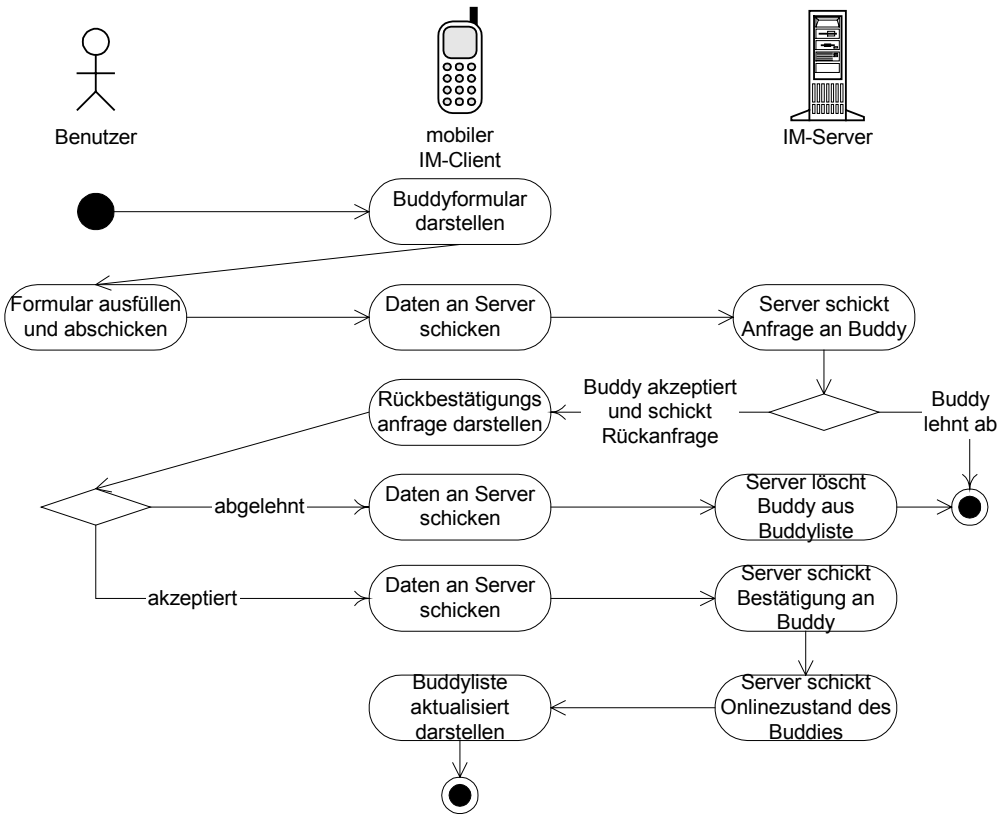

Abbildung 27: Activity-Diagramm des Use-Case Neuen Buddy anlegen

Basic-Flow:

1. Dieser Use-Case startet mit dem Auswählen des Menüpunktes „Neuen Buddy anlegen“ im Buddylist-Screen.
2. Der Benutzer füllt das Formular mit Benutzerkennung und Nickname des Buddys aus und schickt das Formular ab.
3. Der mobile IM-Client baut eine Verbindung zum Server auf und schickt die Anmeldedaten an den Server.
4. Der Server prüft ob eine Anmeldung mit den Daten funktioniert und schickt eine Anwort an den mobilen IM-Client zurück.
5. Dann wird vom IM-Client die neue Buddyliste dargestellt.

Alternate-Flows:

- Falls der Buddy den Subscriptionrequest ablehnt wird dieser automatisch wieder aus der Buddyliste entfernt.
- Falls der Benutzer selbst den Resubscriptionrequest ablehnt wird von beiden Buddylisten der jeweils andere Buddy entfernt.

Preconditions:

- Benutzer muss am Server angemeldet sein.

Postconditions:

- Keine

5.2.4. Use-Case: Buddy löschen

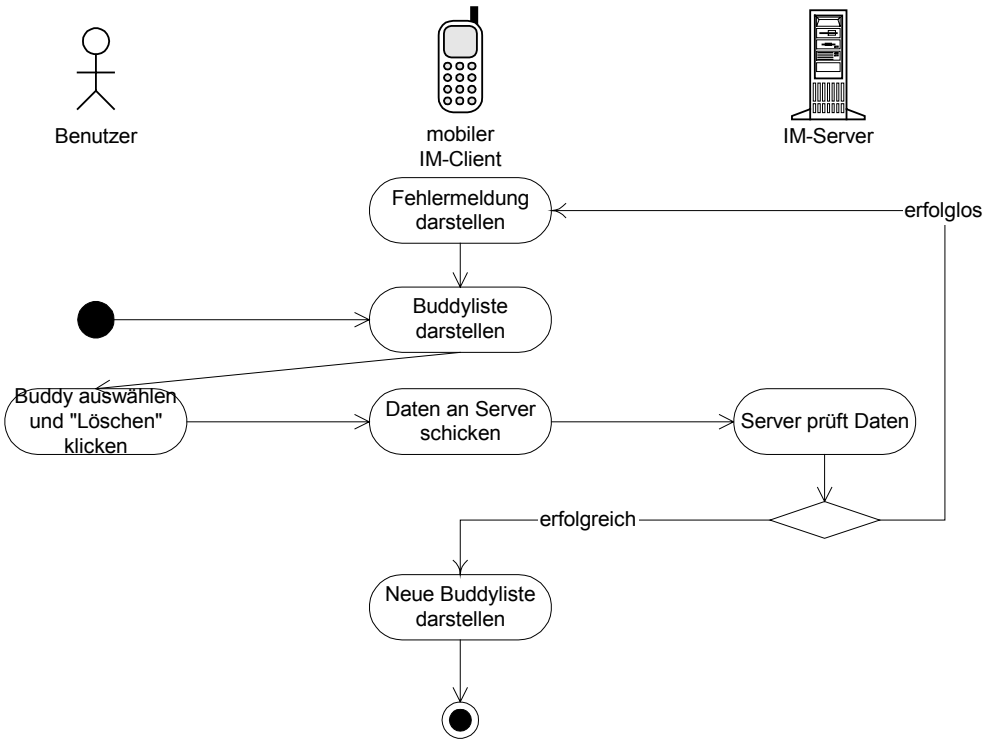

Abbildung 28: Activity-Diagramm des Use-Case Buddy löschen

Basic-Flow:

6. Dieser Use-Case startet mit dem Auswählen des Menüpunktes „Diesen Buddy löschen“ im Buddylist-Screen.
7. Der mobile IM-Client baut eine Verbindung zum Server auf und schickt die Löschdaten an den Server.
8. Dann wird vom IM-Client die neue Buddyliste dargestellt.

Alternate-Flows:

- Keine

Preconditions:

- Benutzer muss am Server angemeldet sein.

Postconditions:

- Keine

5.2.5. Use-Case: Nachricht an Buddy schicken

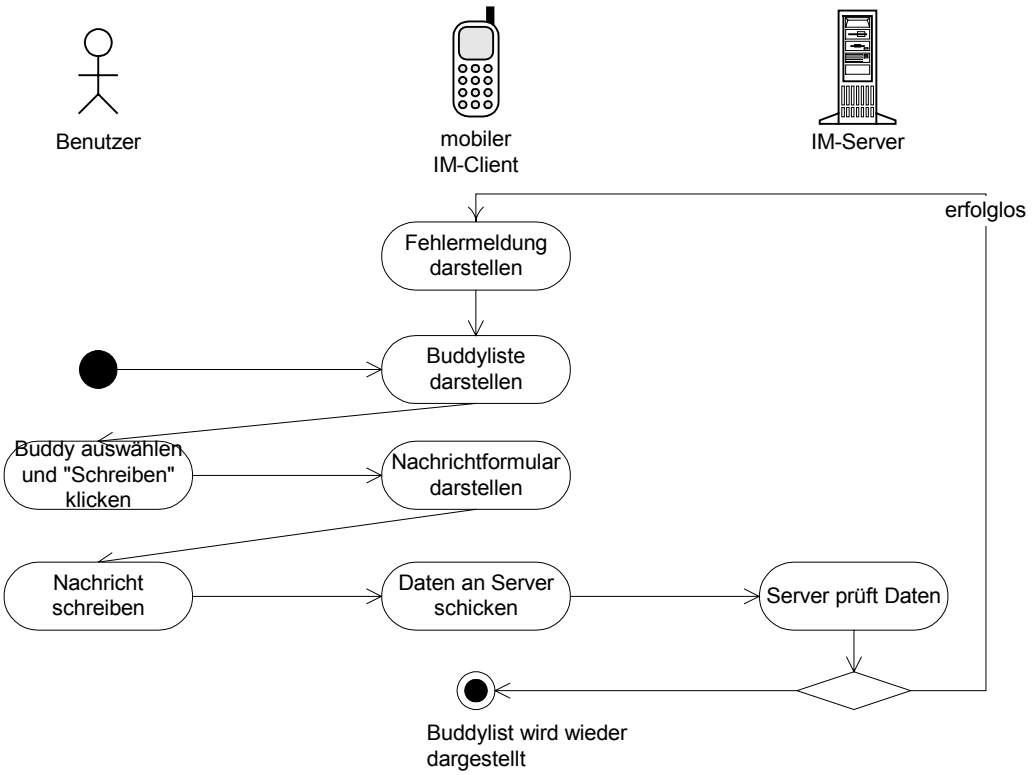

Abbildung 29: Activity-Diagramm des Use-Case Nachricht an Buddy schicken

Basic-Flow:

9. Dieser Use-Case startet mit dem Auswählen des Menüpunktes „Nachricht schreiben“ im Buddylist-Screen.

10. Der Benutzer füllt das Formular aus und wählt den Menüpunkt „Senden“.
11. Der mobile IM-Client baut eine Verbindung zum Server auf und schickt die Nachricht an den Server.
12. Dann wird vom IM-Client wieder die Buddyliste dargestellt.

Alternate-Flows:

- Keine

Preconditions:

- Benutzer muss am Server angemeldet sein.

Postconditions:

- Keine

5.2.6. Use-Case: Onlinezustand ändern

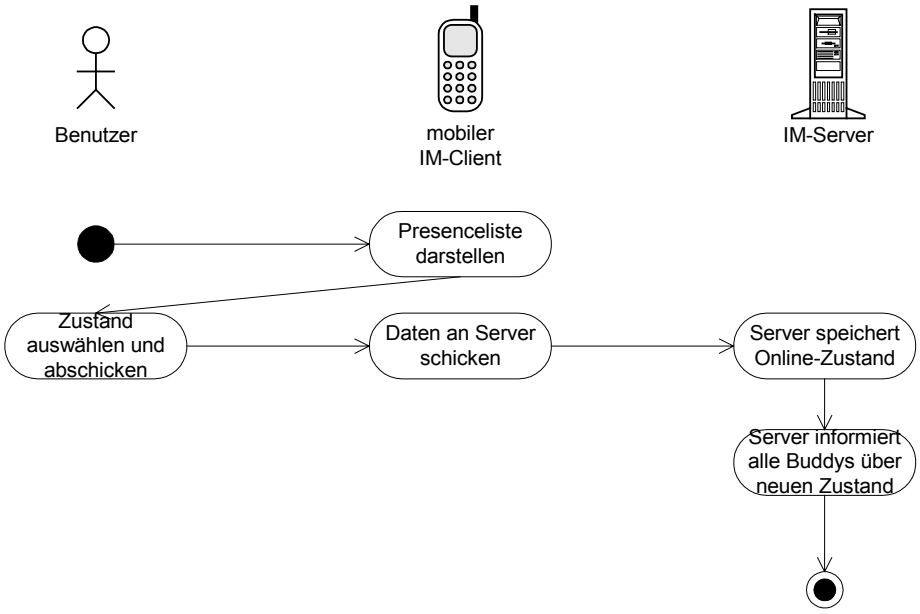

Abbildung 30: Activity-Diagramm des Use-Case Onlinezustand ändern

Basic-Flow:

13. Dieser Use-Case startet bei der Darstellung der Onlinezustände-Liste.
14. Der Benutzer wählt einen Zustand aus und wählt den Menüpunkt „Ok“.
15. Der mobile IM-Client baut eine Verbindung zum Server auf und schickt den neuen Onlinezustand an den Server.
16. Dann wird vom IM-Client wieder die Buddyliste dargestellt.

Alternate-Flows:

- Keine

Preconditions:

- Benutzer muss am Server angemeldet sein.
- Auswählen des Menüpunktes „Onlinestatus“ im Buddylist-Screen.

Postconditions:

- Keine

5.2.7. Use-Case: Informationen zustellen

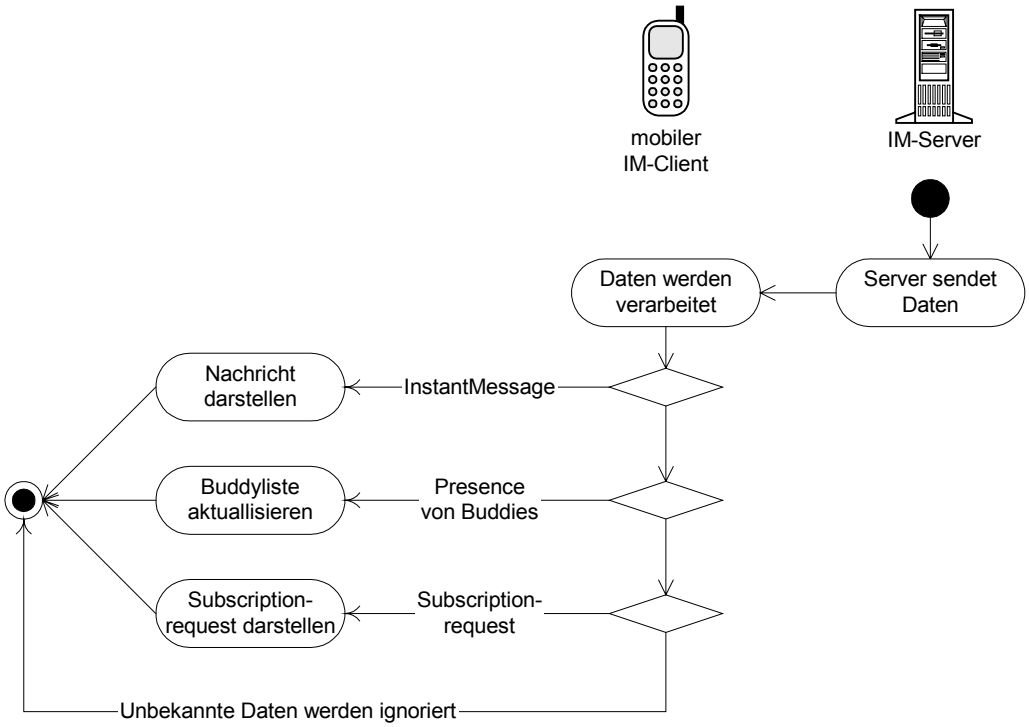

Abbildung 31: Activity-Diagramm des Use-Case Informationen zustellen

Basic-Flow:

1. Der Server schickt Daten an den IM-Client.
2. Die Daten werden vom IM-Client verarbeitet und interpretiert.
3. Je nach Art der Daten wird entweder eine Nachricht dargestellt, die Buddyliste aktualisiert oder ein Subscriptionrequest dargestellt.

Alternate-Flows:

- Keine

Preconditions:

- Benutzer muss am Server angemeldet sein.

Postconditions:

- Keine

5.3. Auswahl der verwendeten Technologien

5.3.1. Das Instant-Messaging-Protokoll Jabber

Das derzeitig am besten entwickelte, frei verfügbare Instant-Messaging-Protokoll ist das XML-basierte OpenSource-Protokoll Jabber. Properitäre Protokolle, wie sie von den bekannten Instant-Messaging-Systemen wie ICQ, AIM, MSM oder Yahoo! verwendet werden sind nicht öffentlich und können daher nur unter sehr großem Aufwand implementiert werden. Ein weiterer Vorteil von Jabber sind die Transport-Dienste (Gateways), mit denen ein Jabber-Benutzer auch mit Benutzern anderen bekannten IM-Systeme kommunizieren kann. Zudem gibt es eine enorme Anzahl an kostenfreien Produkten, wie Server oder Clients für viele verschiedene Betriebssysteme.

Folgende Abbildung soll deutlich machen, dass die IM-Systeme wie ICQ, AIM, MSM oder Yahoo! nur eine Kommunikation innerhalb des Systems zulassen. Der Jabber-Server kann auch mit fremden IM-Systemen kommunizieren. Dabei wird eine Nachricht an eine fremde Adresse an die EtherX-Routerkomponente des Jabber-Servers weitergeleitet. Ist die Nachricht für ein anderes Jabber-System gedacht, wird von EtherX aus die Nachricht an den anderen Jabber-Server direkt weitergereicht. Ist die Nachricht für ein IM-System wie ICQ, AIM, MSM oder Yahoo!, dann wird von EtherX aus die Nachricht an den entsprechenden Transportdients weitergeleitet, der dann verschiedenen Clients simuliert und somit mit jedem dieser Systeme kommunizieren kann.

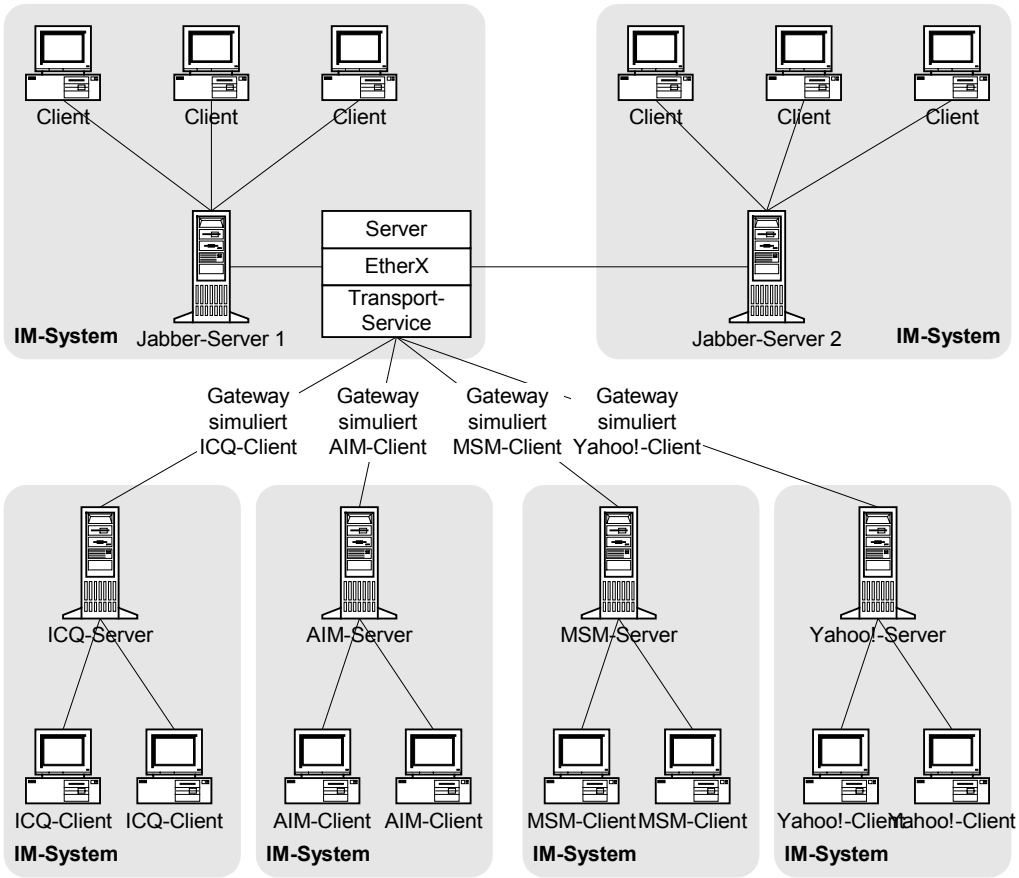

Abbildung 32: Kommunikation der IM-Systeme

Der große Vorteil von XML-Protokollen ist die Lesbarkeit, Flexibilität und die Tatsache, dass es sich bei XML um einen international anerkannten Standard für die Strukturierung von Informationen handelt. Mit einem einfachen Texteditor können XML-Daten erstellt und gelesen werden. Der Nachteil von XML ist der große Overhead, der durch die XML-Tags <tag> entsteht.

Das Anmelden am System geschieht mit folgender Nachricht an den Server:

```
<iq type=”set” id=”1001”>
  <query xmlns=”jabber:iq:auth”>
    <username>hans</username>
    <password>geheim</password>
    <resource>mobil</resource>
  </query>
</iq>
```

Wenn die Anmeldung erfolgreich war schickt der Server folgende Antwort:

```
<iq type=”result” id=”1001” />
```

War zum Beispiel das Passwort falsch, dann wird folgende Antwort geschickt:

```
<iq type=”error” id=”1001”>
  <error code=”400”>Wrong Password</error>
</iq>
```

Der mobile Instant-Messaging-Client bekommt den Namen „mJabber“. Das „m“ steht für mobil und „Jabber“ für das verwendete Protokoll. Ab sofort wird der mobile Instant-Messaging-Client mJabber genannt.

Folgendes Beispiel wird die Funktionsweise des Jabber-Protokolls genauer erläutern. Es ist der Ablauf einer Subscription dargestellt. Eine Subscription ist der Vorgang wenn ein Benutzer einen neuen Buddy in seiner Buddylist anlegen will um dessen Onlinestatus einzusehen. Bei dem Vorgang wird der neu hinzugefügte Buddy um seine Einwilligung gebeten. Akzeptiert dieser die Anfrage, so wird automatisch eine Rückanfrage geschickt. Wenn beide die Anfrage akzeptieren ist der Vorgang abgeschlossen und jeder bekommt automatisch ab diesem Zeitpunkt vom Server die Informationen über den Onlinestatus des anderen zugeschickt.

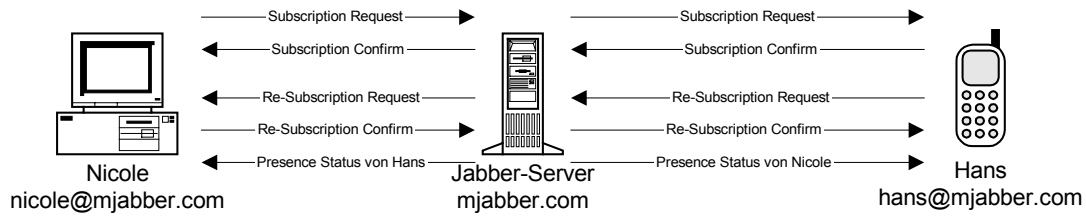

Abbildung 33: Subscription-Ablauf bei Jabber

Tabelle 8: Subscription-Ablauf mit dem Jabber-Protokoll

Nicole möchte Hans in die Buddyliste aufnehmen. Dazu schickt sie einen Subscriptionrequest an den Server: `<presence to="hans@mjabber.com" type="subscribe" />`	

	Hans bekommt einen Nachricht vom Server:
	
	`<presence to="hans@mjabber.com" from="nicole@mjabber.com" type="subscribe" />`
	
	Hans ist damit einverstanden, dass Nicole seinen Onlinezustand sehen kann und akzeptiert ihre Anfrage mit folgender Nachricht an den Server:
	
	`<presence to="nicole@mjabber.com" type="subscribed" />`
	
	Gleichzeitig wird automatisch eine Rückanfrage (Resubscriptionrequest) an Nicole gestellt:
	
	`<presence to="nicole@mjabber.com" type="subscribe" />`

Nicole wird vom Server informiert, dass Hans einverstanden ist:
 	
`<presence to="nicole@mjabber.com" from="hans@mjabber.com" type="subscribed" />`
 	
...und die Anfrage von Hans:
 	
`<presence to="nicole@mjabber.com" from="hans@mjabber.com" type="subscribe" />`
 	
Nicole ist natürlich auch damit einverstanden, dass Hans ihren Onlinezustand stehen kann und akzeptiert seine Anfrage mit folgender Nachricht an den Server:
 	
`<presence to="hans@mjabber.com" type="subscribed" />`	

	Hans wird vom Server informiert, dass Nicole einverstanden ist: `<presence to="hans@mjabber.com" from="nicole@mjabber.com" type="subscribed" />`

Nicole bekommt nun von Hans den aktuellen Online-Status:
 	
`<presence to="nicole@mjabber.com" from="hans@mjabber.com">`
`  <status>Available</status>`
`</presence>`
	Hans bekommt nun von Nicole den aktuellen Online-Status:
	
	`<presence to="hans@mjabber.com" from="nicole@mjabber.com">`
	`  <status>Available</status>`
	`</presence>`

5.3.2. Programmiersprache Java mit J2ME

Mit J2ME ist Java die optimale Programmiersprache für den mobilen Instant-Messaging-Client, weil diese Programmiersprache zum einen von sehr vielen Mobiltelefonen unterstützt wird und zum anderen die Möglichkeit bietet das XML-basierte Jabber-Protokoll zu interpretieren oder zu erstellen. Zudem kann man mit J2ME eine ansprechende und intelligente Benutzeroberfläche erstellen.

Eine browserbasierte Lösung wäre auch möglich, wenn ein zusätzlicher Webserver die Interpretation des Protokolls übernehmen würde und diese dann als WML-Seiten darstellen würde. Aber diese Lösung hätte viele Nachteile, da Nachrichten, Online-Informationen der Buddys usw. immer vom Benutzer selbst abgefragt werden müssen und nicht vom Server zugestellt werden könnten. Dies würde einen zusätzlichen Traffic und somit auch höhere Kosten bedeuten.

5.3.3. Verbindung via GPRS

Empfehlenswert ist die Einwahl des Mobiltelefons via GPRS, da somit die Kosten nur nach Traffic und nicht nach Onlinezeit berechnet werden und der Benutzer dann ständig mit dem Server verbunden bleiben kann. Die Übertragungsgeschwindigkeit von GPRS reicht vollkommen für den Datentransfer aus, da hier immer nur wenige Daten verschickt werden.

5.4. Design

Wegen der Komplexität des Clients wurde die Aufteilung in Model, View und Controller gewählt. Somit ist jeder dieser Komponenten für einen klar abgeteilten Aufgabenbereich verantwortlich. Der Controller ist für die Logik, die Netzwerkverbindung und für die Interpretation des Protokolls zuständig. Das Model verwaltet alle Daten der Anwendung und der View steuert die Darstellung der Informationen auf dem Display des Mobiltelefons. Diese Aufteilung hat den Vorteil, dass Ergänzungen, Erweiterungen oder Portierung auf ein anderes Gerät oder Protokoll schneller vorgenommen werden können.

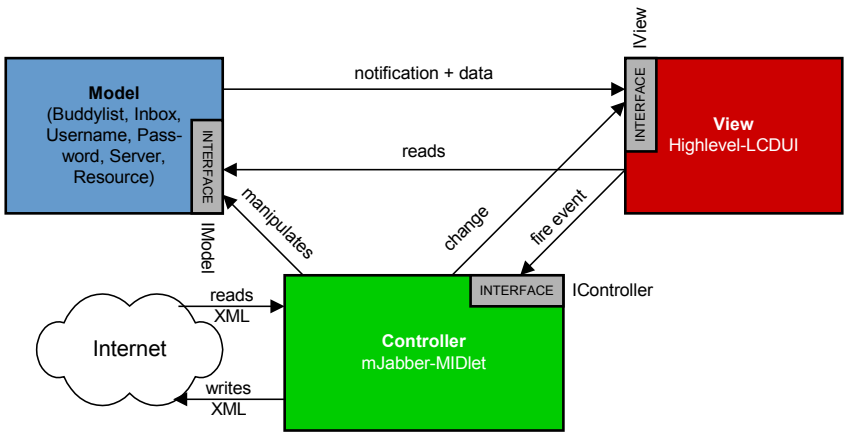

Abbildung 34: Model-View-Controller Architektur bei mJabber

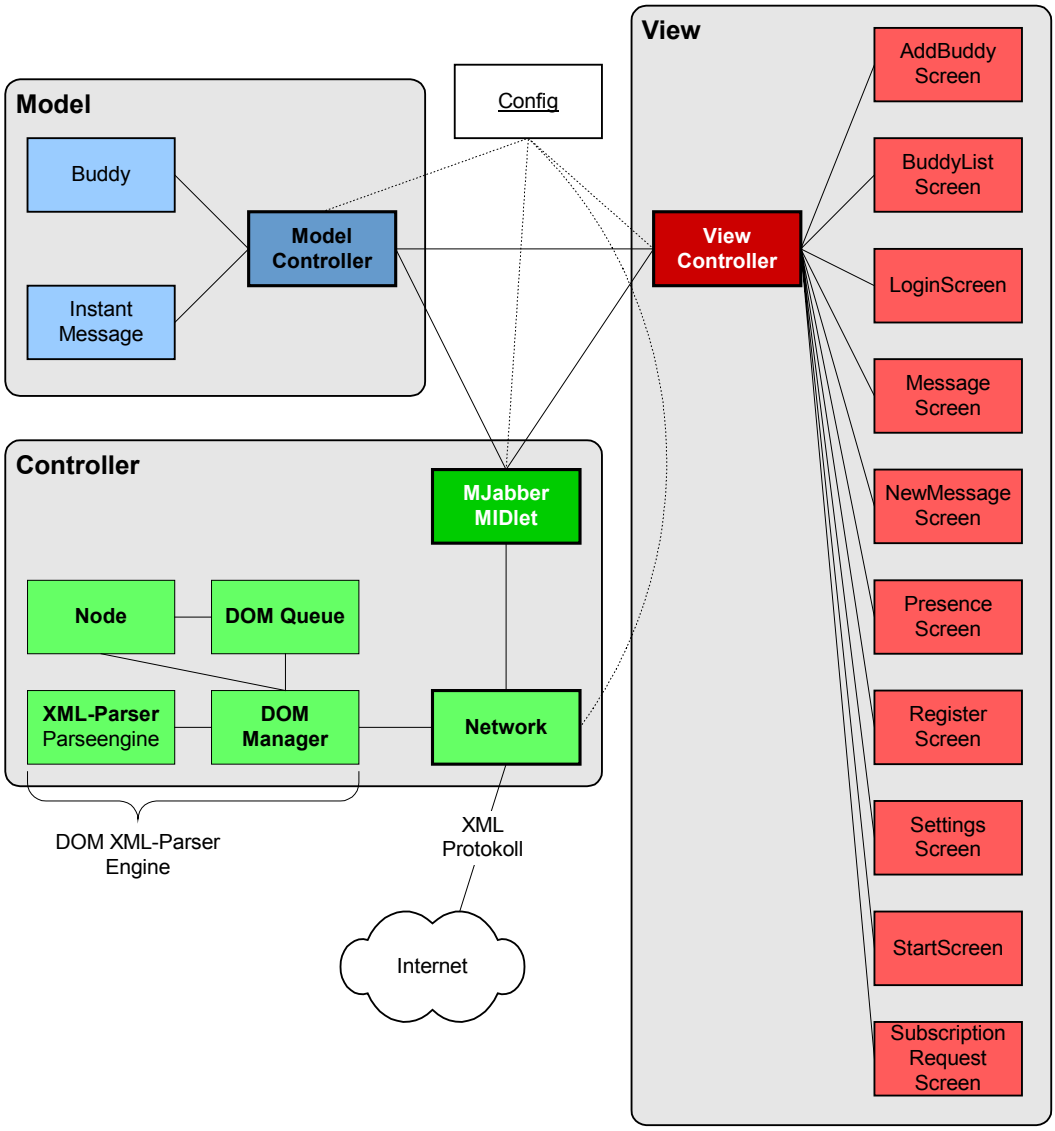

Abbildung 35: Klassendiagramm von mJabber

Am folgenden Beispiel wird die Funktionsweise des Model-View-Controller-Prinzips, wie ich es in mJabber implementiert habe, erläutert. Im diesem Beispiel empfängt mJabber eine Nachricht von einem anderen Jabber-Benutzer.

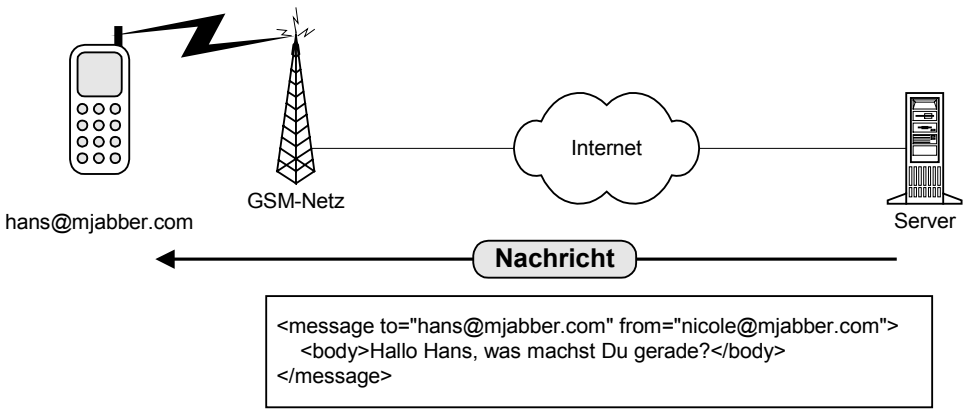

Abbildung 36: mJabber empfängt eine Nachricht

Die Funktionsweise des XML-Parsers wird später genauer beschrieben, daher wird hier das Parsen des Jabber-Protokolls in einem Punkt zusammengefasst.

Die Network-Klasse empfängt den XML-Datenstrom. Dieser Datenstrom wird an den XML-Parser weitergeleitet. Dieser macht daraus eine XML-Datenstruktur (DOM) und gibt diese dann an die Network-Klasse zurück. Diese interpretiert die Struktur und ruft dann die entsprechende Methode in der Hauptcontroller-Klasse MJabber auf. Die MJabber-Klasse informiert die Model-Klasse, dass eine neue Nachricht eingetroffen ist. Diese ist die Schnittstelle zwischen Controller und Model („Controller manipulates Model“).

Das Model legt ein neues InstantMessage-Objekt an und speichert dies in einem Vector (Collection) ab. Dann wird die View-Klasse informiert, dass eine neue Nachricht eingetroffen ist. Dieser Vorgang ist die Schnittstelle zwischen Model und View („Model notificates View“). Die View-Klasse prüft welcher Screen gerade dargestellt wird. Wird gerade eine Nachricht geschrieben oder eine Nachricht gelesen wird die neue Nachricht noch nicht dargestellt. Wird allerdings die Buddyliste dargestellt, so holt die View-Klasse von der Model-Klasse die nächste Instant-Message („View reads from Model“), damit wird dann der MessageScreen aktualisiert und dann dargestellt.

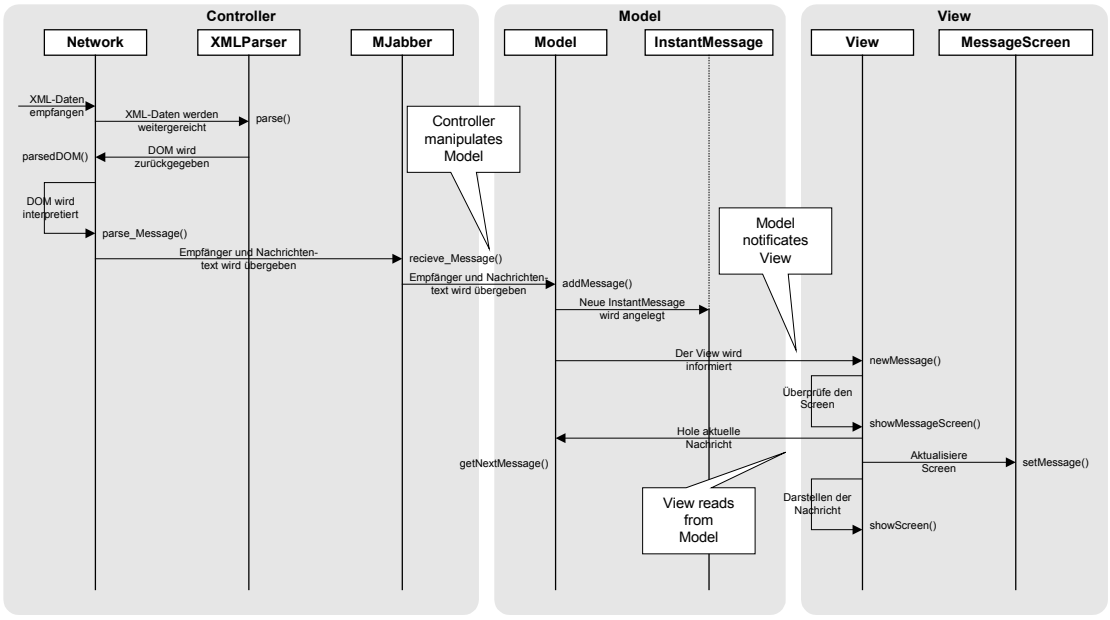

Abbildung 37: Sequenz-Diagramm: Eingang einer Nachricht

Für eine flexible Sprachgestaltung des Clients wurden alle Texte, die innerhalb des Programms erscheinen, in eine statische Config-Klasse geschrieben. So kann in kürzester Zeit mJabber für viele Sprachen angeboten werden. Außerdem besteht die Möglichkeit durch einen Start-Screen mit einer Grafik mJabber zum Beispiel für Mobilfunkprovider grafisch anzupassen. Somit könnte der IM-Client von Vodafone, T-Mobile, eplus oder O2 unter eigenem Namen angeboten werden.

5.5. Der XML-Parser

Eines der größten Problem, die es zu Lösen galt, war ein XML-Parser, der das Jabber-Protokoll, welches vom Server geschickt wird, bearbeitet. Dazu wurde der XML-Parser so konstruiert, dass er den Datenstrom vom Server nach den XML-Tags <tag> durchsucht und dann entsprechende Methoden im XMLEventListener-Interface aufruft. Es gibt drei verschiedene Methoden im XMLEventListener-Interface: Eine für Start-Tags, eine für den reinen Textinhalt und eine für Ende-Tags. Diesen Teil des Parsers nennt man „Eventgesteuerter XML-Parser“.

Der DOM-Manager implementiert das XMLEventListener-Interface. Damit kann er, gesteuert vom XML-Parser, unter Verwendung der Node-Klasse eine navigierbare Datenstruktur (DOM) erstellen. Die Node-Klasse repräsentiert ein XML-Element und somit auch einen Knotenpunkt in der DOM-Struktur. Sobald die DOM-Struktur fertig ist, wird diese dann in eine Queue (Warteschleife) gestellt.

Die Queue schickt dann in regelmäßigen Abständen die fertigen Strukturen an das DOM-EventListener-Interface.

Bei mJabber hat die Network-Klasse das DOM-EventListener Interface implementiert. Diese Klasse interpretiert dann die fertige Datenstruktur und ruft je nachdem verschiedene Methoden in der Controller-Klasse mJabber auf. Darauf hin aktualisiert der Controller dann das Datenmodel.

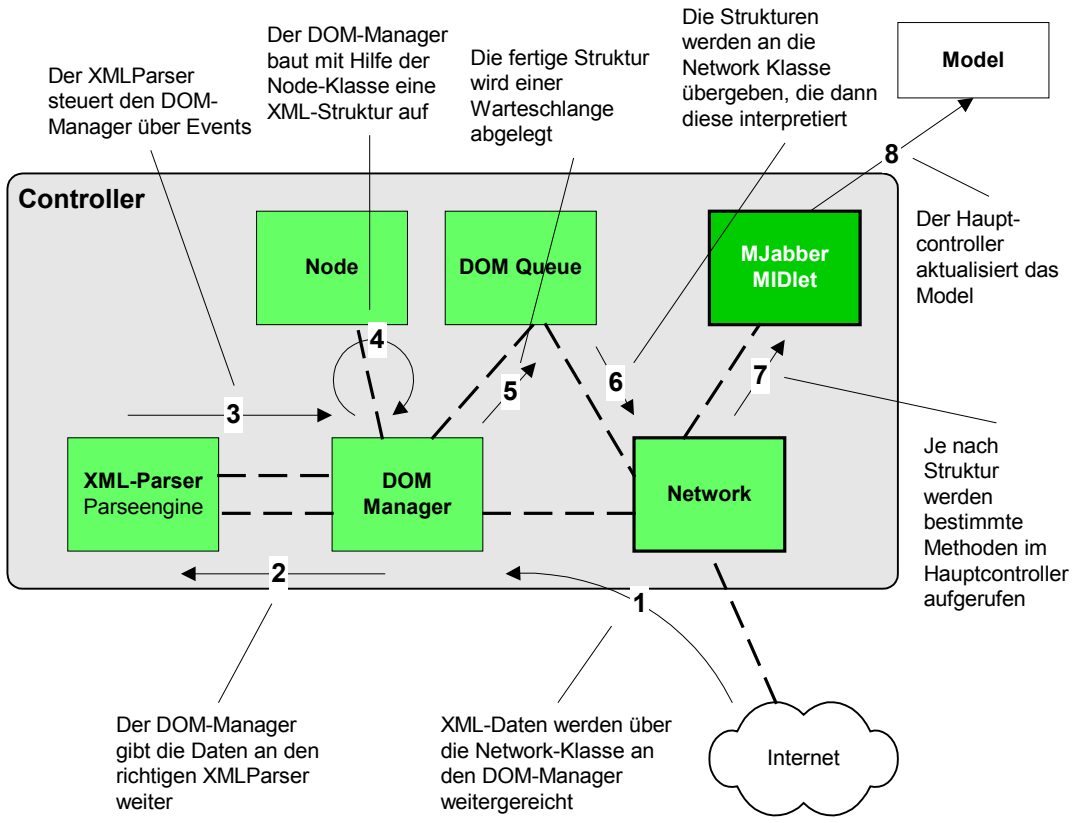

Abbildung 38: Funktionsweise des Parsen von XML Daten

5.6. Implementierung

Bei der Implementierung des mobilen IM-Clients habe ich einen Texteditor[18] und das Wireless Toolkit (WTK) von SUN verwendet (siehe Kapitel 4.5.). Zur Erstellung des mJabber MIDlets bin ich in folgenden fünf Schritten vorgegangen:

1. Erstellen kleiner MIDlets um die Funktionsweise der grafischen Benutzerschnittstelle (GUI), des Generic Connection Frameworks (GCF) und des Record Management Systems (RMS) näher kennen zu lernen.
2. Erstellen eines XMLParsers, der später dann das Jabber-Protokoll parsen soll.
3. Erstellen eines ganz simplen IM-Clients, der sich nur einloggen und eine Nachricht verschicken kann. Den korrekten Versand der Nachricht konnte ich mit einem zweiten Jabber-Account und dem Windows-Jabber-Client WinJab[19] testen. Dieser Jabber-Client verfügt über die Möglichkeit das XML-Protokoll direkt anzuzeigen.
4. Erstellen eines Use-Case-Diagramms, eines Klassen-Diagramms, verschiedene Activity-Diagrammen und vielen Skizzen für die Bildschirmdarstellung, verschiedene Icons und die Navigation.
5. Erstellen des richtigen IM-Clients mJabber mit den Funktionen, wie sie in den Use-Case-Diagrammen beschrieben wurden.

Durch diese Vorgehensweise war es mit möglich innerhalb der kurzer Zeit von zwei Monaten den funktionstüchtigen mobilen IM-Client mJabber zu entwickeln.

5.7. Ähnliche Produkte auf dem Markt

Zum Anfang der Diplomarbeit gab es nur sehr wenig ähnliche Produkte auf dem Markt. Die Firma µppli[20] brachte mit uMessenger und uMessenger plus einen der ersten J2ME-fähigen mobilen Jabber Client auf den Markt. Nach vier Monaten Bearbeitungszeit dieser Arbeit sind jetzt schon einige mobile Instant-Messaging-Clients auf dem Markt verfügbar.

Tabelle 9: Derzeit verfügbare J2ME-Instant-Messaging-Clients

Name	IM-System	Internetseite
uMessanger und uMessanger plus	Jabber	www.uppli.com
ACCEPT	Jabber	www.antepo.com
TipicME	Jabber	www.tipic.com
Siemens M50 Special Edition	ICQ	www.wirelesssoftware.info/show1news.php/248.html
SIM	Jabber	www.streampath.com/Sim_jabber.htm

[18] Code Genie; http://www.code-genie.com/

[19] WinJab; http://winjab.sourceforge.net/

[20] µppli, http://www.uppli.com

6. Fazit

Kommunikationsdienste wie mobile IM-Systeme können und werden sehr wahrscheinlich an die großen Erfolge von SMS anknüpfen, sobald die Preise für die dafür notwendigen mobilen Datenverbindungen etwas günstiger werden. Die großen Vorteile von IM-Systemen im Vergleich zu SMS sind die zusätzlichen Dienste, wie zum Beispiel die Information ob der Gesprächspartner gerade Zeit hat oder die Möglichkeit Dateien, ohne zusätzliche Software, direkt mit einem anderen Endgerät auszutauschen. Aber J2ME wird nicht nur für mobile IM-Clients verwendet werden.

Was kann man mit J2ME noch machen? Auf diese Frage ist es nicht einfach eine kurze Anwort zu finden, da die Möglichkeiten enorm sind. Stellt man sich mal vor, dass in ein paar Jahren jeder ein javafähiges Mobiltelefon besitzt, dann wird klar welche Rolle die Softwareentwicklung für diese Geräte spielen wird. Hunderte Millionen von Endgeräte mit denen man Steuern, Kommunizieren und noch vieles mehr machen kann.

Zucotto Systems Inc. hat als erste Firma die KVM sogar schon in Hardware realisiert.[21] Damit werden J2ME Anwendungen um vieles schneller und leistungsfähiger. Dieser Chip heißt Xpresso und unterstützt J2ME und die CLDC komplett. Endgerätehersteller können diesen Chip einbauen um so hochperformante J2ME-Endgeräte herzustellen.

Nachfolgend nenne ich einige Beispiele, die zeigen, was mobile Applikationen heute alles schon leisten könnten:

- Steuerung der Haustechnik mit dem Mobiltelefon: Es klingelt an der Türe und auf dem Mobiltelefon erscheint ein Bild von der Person vor der Türe. Man kann nun mit dieser Person (über das Mobiltelefon) reden und die Türe öffnen. Selbst Fernseher, Videorecorder, Stereoanlage usw. lassen sich dann vom Mobiltelefon als universale Fernbedienung aus steuern. Die Heizung, auch die Rolländen sowie die Waschmaschine oder die Kaffeemaschine könnten dann gesteuert werden.
- Das Mobiltelefon für das Profil-Management: Man betritt einen Raum indem eine Anlage über Bluetooth sofort mit dem Mobiltelefon kommuniziert. Die Anlage erkennt das Mobiltelefon und stellt die Raumtemperatur, das Licht und die Musik auf die entsprechende Person ein.
- Das Mobiltelefon als Dating-Hilfe: Eine Software wird auf dem Mobiltelefon installiert. In der Software legt man ein persönliches Profil und die Bedürfnisse und Wünsche fest. Sobald nun das Mobiltelefon in die Nähe eines anderen Mobiltelefons mit der gleichen Software kommt werden die Profile automatisch verglichen und bei Übereinstimmung bekommen beide Mobiltelefonbesitzer ein Alarmsignal.

- Das Mobiltelefon als mobiles und elektronisches Ticket: Auf dem Mobiltelefon könnte per SMS ein eindeutiger, fälschungssicher codierter Zahlencode (verschlüsselte Ticketnummer) hinterlegt werden. An der Kasse wird dann über Bluetooth das Ticket abgerufen und bestätigt.

Das waren nur ein paar Ideen, was mit J2ME heute schon alles möglich wäre. In Zukunft werden die Endgeräte immer leistungsfähiger und haben mehr Funktionalitäten (zum Beispiel eine Kamera). Das Spektrum der Möglichen mobilen Anwendungen wird immer größer werden. Sobald die 3D-Graphics-API von den ersten Herstellern implementiert wird werden die mobilen Spiele eine neue Ära einleiten. Nokia entwickelt derzeit das erste Gamephone („Gameboy“ und Mobiltelefon in einem) unter dem Namen N-Gage.[22] Ich vermute, dass die Entwicklung der verschiedenen mobilen Endgeräte hin zu einem Universalgerät gehen werden, welches für alle Zwecke des Alltags geschaffen ist. Ein Mobiltelefon, mit den Eigenschaften eines PDAs, mit dem man kommunizieren, unterwegs arbeiten, steuern, sich unterhalten lassen (mit Musik, Video und Spiele), fotografieren und vielleicht sogar Türen öffnen und damit bezahlen kann. Nokia hat mit dem 7650[23] und Sony-Ericcson mit dem P800[24] die ersten Geräte in dieser Art entwickelt.

Abbildung 39: Die zukünftigen Mobiltelefone: N-Gage , das 7650 und das P800

Nachdem jetzt nahezu jedes Mobiltelefon Java mit J2ME unterstützt wird dieser Technologie sehr wahrscheinlich ein großer Erfolg bevorstehen. Zudem könnte J2ME eine Schlüsseltechnologie für mobile Bluetooth-Anwendungen werden. Kleine Terminals an Einkaufswagen könnten stets die aktuellen Angebote der umliegenden Regale anzeigen und bei Fragen nach Produkten den schnellsten Weg zum entsprechenden Regal zeigen. Wir dürfen gespannt sein, was uns in den nächsten Monaten und Jahren erwarten wird. Der Phantasie sei keine Grenzen gesetzt.

---

[21] [KVM]
[22] [INFOSYNC]
[23] [NOKIA]
[24] [SONY]

Weiterführende Internetadressen

SUN's offizielle J2ME-Seite:

- http://java.sun.com/j2me/

J2ME-fähige Endgeräte:

- http://wireless.java.sun.com/device/
- http://www.javamobiles.com/
- http://www.microjava.com/devices/
- http://kissen.cs.uni-dortmund.de:8080/devicedb/

J2ME-Developer Communities:

- http://wireless.java.sun.com/
- http://www.microjava.com/
- http://www.midletcentral.com/
- http://www.linecity.de/
- http://www.midlet.org/

Jabber-Internetseiten:

- http://www.jabber.org/ (Jabber Software Foundation)
- http://www.jabber.org/protocol/ (Jabber Protokoll Definition)
- http://www.jabber.com/
- http://www.jabberstudio.org/
- http://www.jabbercentral.org/
- http://www.jabberview.com/ (Übersicht der öffentlichen Jabber-Server)
- http://www.jabberpowered.org/ (Logos für Jabber-fähige Software)

Glossar

Buddy: Ein Buddy ist der Eintrag eines anderen Benutzers des IM-Systems, welchen man in der sogenannten Buddylist aufgenommen hat.

Buddylist: Die Buddylist ist das Adressbuch in einem IM-System.

Client-Server: Client-Server-Systeme, wie der Name schon sagt bestehen aus Clients und Server. Dabei ist die Kommunikation immer nur zwischen einem Client und einem Server. Die momentan populärste Client-Server Anwendung ist das WWW.

i-mode: Ein cHTML und GPRS-basierter Service ähnlich dem WAP. Im asiatischen Raum ist dieser Service sehr populär. In Deutschland wir er derzeit nur von eplus vertrieben.

Jabber: Offenes XML-basiertes Instant-Messaging-Protokoll. Jabber wurde 1998 von Jeremie Miller erfunden und bis heute ständig weiterentwickelt.

J2ME: Java 2 Mirco Edition. Neben der Standard- und Enterprise-Edition (J2SE und J2EE) ist die Micro Edition das passende Java-Framework für mobile Endgeräte wie Mobiltelefone.

MIDlet: Analog zum Applet ist das MIDlet die Anwendung, die auf einem Endgerät mit MIDP/CLDC/KVM läuft.

Peer-to-Peer: Eine Peer-to-Peer-Verbindung ist ein Netzwerkverbindung zwischen zwei Endgeräten (direkte Kommunikation).

Stand-Alone: Eine Stand-Alone-Anwendung ist unabhängig von einem Netzwerk, einem Server oder einem anderen Rechner. Die Anwendung kann „alleine“ betrieben werden.

VM: Virtual Machine, die den Java Bytecode interpretiert.

VoIP: Voice over IP ist ein Vorgang bei dem Sprachdaten über ein IP-Netz verschickt werden.

Literaturverzeichnis

[MIDP] C. Enrique Ortiz, Eric Giguère (2001): Mobile Information Device Profile for Java 2 Mirco Edition; Wiley, Professional Developer's Guide Series; ISBN 0-471-03465-7

[J2ME] Eric Giguère (2002): Java 2 Mirco Edition; Wiley, Professional Developer's Guide Series; ISBN 0-471-39065-8

[BHV] Sebastian Eschweiler (2003): Das Einsteigerseminar Java 2 Mirco Edition; bhv; ISBN 3-8266-7248-8

[OR] Kim Topley (2002): J2ME in a Nutshell; O'Reilly; ISBN 0-596-00253-X

[RIGGS] Roger Riggs u.a. (2001): Programming Wireless Devices with the Java 2 Platform, Micro Edition; Addison-Wesley; ISBN 0-201-74627-1

[JS] Alfred Vockinger (2002): Java to go: Java auf Mobilgeräten; JavaSpektrum 7/8/2002 (Seite 20 – 25)

[WML] WapForum (2001): Wireless Markup Language Version 2.0; http://www1.wapforum.org/tech/documents/WAP-238-WML-20010911-a.pdf (Datum des Zugriffs:)

[CHTML] Tomihisa Kamada (09.02.1998): Compact HTML for Small Information Appliances; W3C; http://www.w3c.org/TR/1988/NOTE-compactHTML-19980209/ (Datum des Zugriffs: 12.12.2003)

[IMODE] Martin Didier (20.09.2000): Getting into i-Mode; XML.com; http://www.xml.com/pub/a/2000/09/20/wireless/imode.html (Datum des Zugriffs: 12.12.2002)

[ECIN] Ralf Koyro u.a. (23.05.2002): Trendbericht: Technik in der Telekommunikation; ECIN; http://www.ecin.de/technik/standards/ (Datum des Zugriffs: 12.12.2002)

[GSM] GSM World: http://www.gsmworld.com/

[CHIP] Michael Brunn (01.2002): Kabel ade; CHIP Online; http://www.chip.de/produkte_tests/produkte_tests_8615275.html (Datum des Zugriffs 16.01.2003)

[SMS] Florian Rötzer (01.10.2000): SMS erfreut sich schnell wachsender Beliebtheit; http://www.telepolis.de/deutsch/inhalt/te/8826/1.html (Datum des Zugriffs: 11.01.2003)

[FTD] Financial Times Deutschland (09.10.2002, 15:04):NTT Docomo testet UMTS-Nachfolger; http://www.ftd.de/tm/tk/1034160663547.html?nv=rs (Datum des Zugriffs: 13.01.2003)

[ARTEM] AR*tem*)))) (11.2002):Neue Wireless LAN Technologien; http://www.artem.de/download/dokumente/white-54mbs.pdf (Datum des Zugriffs: 21.01.2003)

[HEISE1] Heise Newsticker (08.01.2003, 12:32): Funknetze: HomeRF-Arbeitsgruppe löst sich auf; ea@ct.heise.de; http://www.heise.de/newsticker/data/ea-08.01.03-000/

[HEISE2] Heise Newsticker (26.11.2002, 16:47): Schnurlos surfen in Coffeebars; dz@ct.heise.de; http://www.heise.de/newsticker/data/dz-26.11.02-000/

[SUN] SUN Mircosystems: Java 2 Micro Edition; http://java.sun.com/j2me/ und SUN Wireless Developer Homepage; http://wireless.java.sun.com/

[WBXML] Bruce Martin, Bashar Jano (24.06.1999): WAP Binary XML Content Format; W3C; http://www.w3c.org/TR/wbxml/ (Datum des Zugriffs: 18.01.2003)

[SERIALIZATION] Eric Giguère (27.02.2002): Object Serialization in CLDC-Based Profiles; SUN Wireless Developer Homepage – Techtips; http://wireless.java.sun.com/midp/ttips/serialization/ (Datum des Zugriffs: 18.01.2003)

[JABBER] Jabber Software Foundation: http://www.jabber.org/

[KVM] Angus Muir (10.10.2000): Zucotto – Embedding the KVM in Hardware; Micro Java Network; http://www.microjava.com/articles/reviews/zucotto/

[INFOSYNC] Jørgen Sundgot (05.11.2002): Wireless game console from Nokia; infoSync; http://www.infosync.no/news/2002/n/2548.html (Datum des Zugriffs: 18.01.2003)

[NOKIA] Nokia: Informationsseite zum Mobiltelefon 7650; http://www.nokia.com/phones/7650/ (Datum des Zugriffs: 18.01.2003)

[SONY] Sony Ericcson: Informationsseite zum Mobiltelefon P800; http://www.sonyericcson.com/P800/ (Datum des Zugriffs: 18.01.2003)

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift